

# A reliable Turing machine

Ilir Çapuni

Peter Gács

March 9, 2018

## Abstract

The title says it.

## Contents

1	Introduction . . . . .	2
1.1	To be written . . . . .	2
1.2	Turing machines . . . . .	2
1.3	Codes, and the result . . . . .	4
1.4	A shortcut solution . . . . .	4
2	Overview of the construction . . . . .	5
2.1	Isolated bursts of faults . . . . .	5
2.2	Hierarchical construction . . . . .	6
2.3	From combinatorial to probabilistic noise . . . . .	8
2.4	Difficulties . . . . .	8
3	Notation . . . . .	11
4	Specifying a Turing machine . . . . .	12
4.1	Universal Turing machine . . . . .	13
4.2	Rule language . . . . .	14
5	Exploiting structure in the noise . . . . .	18
5.1	Sparsity . . . . .	18
5.2	Error-correcting code . . . . .	20

6	The model . . . . .	21
6.1	Generalized Turing machine . . . . .	21
6.2	Simulation . . . . .	24
6.3	Hierarchical codes . . . . .	25
6.4	Trajectories . . . . .	26
7	Simulation structure . . . . .	29
7.1	Head movement . . . . .	30
7.2	Computation phase . . . . .	33
7.3	Forced self-simulation . . . . .	34
7.4	Transfer phase . . . . .	36
8	Health . . . . .	38
9	Healing and rebuilding . . . . .	41
9.1	Annotation, admissibility, stitching . . . . .	41
9.2	The healing procedure . . . . .	44
9.3	Rebuilding . . . . .	46
10	Scale-up . . . . .	48
10.1	Annotated history . . . . .	48
10.2	Super-health . . . . .	50
10.3	The simulation codes . . . . .	51
11	Isolated bursts . . . . .	52
12	Cleaning . . . . .	55
12.1	Cleaning the current level . . . . .	55
12.2	Escape . . . . .	57
12.3	Pass cleaning . . . . .	61
12.4	Attack cleaning . . . . .	63
13	After a large burst . . . . .	65
13.1	Spill bound . . . . .	65

## 1 Introduction

### 1.1 To be written

### 1.2 Turing machines

Our contribution could use any one of the standard definitions of a Turing machine, but it will be convenient to modify it. Let us recall that a Turing machine is defined by a finite set  $\Gamma$  of *internal states*, a finite alphabet  $\Sigma$  of *tape symbols*, a transition function

$$\delta: \Sigma \times \Gamma \rightarrow \Sigma \times \Gamma \times \{-1, 1\},$$

and possibly some distinguished states and tape symbols. Any array  $A \in \Sigma^{\mathbb{Z}}$  is called the *tape configuration*, or *tape content*. At any time, the head is at some integer position  $h$ , and is observing the tape symbol  $A(h)$ . The meaning of  $\delta(a, q) = (a', q', d)$  is that if  $A(h) = a$  and the state is  $q$  then the  $A(h)$  will be rewritten as  $a'$  and  $h$  will change to  $h + d$ .

We will use a model that is slightly different, but has clearly the same expressing power. There is no set of internal states, but the head observes and modifies *two* neighboring tape cells at a time. Thus, a Turing machine is defined as a pair

$$(\Sigma, \tau).$$

The tape alphabet  $\Sigma$  contains at least the distinguished symbols  $\sqcup, 0, 1$  where  $\sqcup$  is called the *blank symbol*. We have a *transition function*

$$\tau: \Sigma^2 \rightarrow \Sigma^2 \times \{-1, 1\}.$$

The tape is blank at all but finitely many positions. A *configuration* is a pair

$$(A, h),$$

$h \in \mathbb{Z}$  is the current *head position*, or *observed cell*, or *current cell*, and  $A \in \Sigma^{\mathbb{Z}}$  is the *tape content*, or *tape configuration*: at position  $p$ , the tape contains the symbol  $A(p)$ . If  $\xi = (A, h)$  is a configuration then we will write

$$\xi.\text{tape} = A, \quad \xi.\text{pos} = h. \tag{1.1}$$

Though the tape alphabet may contain non-binary symbols, we will restrict input and output to binary.

The transition function  $\tau$  tells us how to compute the next configuration from the present one. When the head observes the pair of tape cells with content  $\mathbf{a} = (a_0, a_1)$  at positions  $h, h + 1$  then denoting

$$(\mathbf{a}', j) = \tau(\mathbf{a}),$$

will change tape content at positions  $h, h + 1$  to  $a'_0, a'_1$ , and move the head to tape position to  $h + j$ .

**Definition 1.1** (Fault) We say that a *fault* occurs at time  $t$  if the output  $(\mathbf{a}', j)$  of the transition function at this time is replaced with some other value (which is then used to compute the next configuration).  $\dashv$

### 1.3 Codes, and the result

For fault-tolerant computation, some redundant coding of the information is needed.

**Definition 1.2** (Codes) Let  $\Sigma_1, \Sigma_2$  be two finite alphabets. A *block code* is given by a positive integer  $Q$ —called the *block size*—and a pair of functions

$$\psi_* : \Sigma_2 \rightarrow \Sigma_1^Q, \quad \psi^* : \Sigma_1^Q \rightarrow \Sigma_2$$

with the property  $\psi^*(\psi_*(x)) = x$ . It is extended to strings by encoding each letter individually:  $\psi_*(x_1, \dots, x_n) = \psi_*(x_1) \cdots \psi_*(x_n)$ .  $\dashv$

For ease of spelling out a result, let us consider only computations whose outcome is a single symbol, at tape position 1. We will consider the machine *halted* if it writes the blank symbol at tape position 0.

**Theorem 1** *There is a Turing machine  $M_1$  with a function  $a \mapsto a.output$  defined on its alphabet, such that for any Turing machine  $G$  with alphabet  $\Sigma$  there are  $0 \leq \varepsilon < 1$  and  $\alpha_1, \alpha_2 > 0$  with the following property.*

*For each input length  $n = |x|$  a block code  $(\varphi_*, \varphi^*)$  of block size  $Q = O((\log n)^{\alpha_1})$  can be constructed such that the following holds.*

*Let  $M_1$  start its work from the initial tape configuration  $\xi = (\varphi_*(x), 0)$ . Assume further that during its operation, faults occur independently at random with probabilities  $\leq \varepsilon$ .*

*Suppose that on input  $x$  machine  $G$  at time  $t$  writes the blank symbol at position 0, and value  $y$  at position 1. Then denoting by  $\eta(u)$  the configuration machine  $M_1$  at time  $u$ , at any time  $t'$  after*

$$t \cdot (\log t)^{\alpha_2},$$

*we have  $\eta(t').tape(1).output = y$  with probability at least  $1 - O(\varepsilon)$ .*

We emphasize that the actual code  $\varphi$  of the construction will depend on  $n$  only in a simple way: it will be the “concatenation” of one and the same fixed-size code with itself,  $O(\log \log n)$  times.

### 1.4 A shortcut solution

A fault-tolerant one-dimensional cellular automaton is constructed in [4]. If our Turing machine could just simulate such an automaton, it would become fault-tolerant. This can indeed almost be done provided that the size of the computation is known in advance. The cellular automaton can be made finite, and we

could define a “kind of” Turing machine with a *circular tape* simulating it. But this solution requires input size-dependent hardware.

It seems difficult to define a fault-tolerant sweeping behavior on a regular Turing machine needed to simulate cellular automaton, without recreating an entire hierarchical construction—as we are doing here.

## 2 Overview of the construction

A Turing machine that simulates “reliably” any other Turing machine even when it is subjected to isolated bursts of faults of constant size, is given in [1]. By *reliably* we mean that the simulated computation can be decoded from the history of the simulating machine despite occasional damages.

### 2.1 Isolated bursts of faults

Let us give a brief overview of a machine  $M_1$  that can withstand isolated bursts of faults, as most of its construction will be reused in the probabilistic setting.

We break up the task of error correction into several problems to be solved. The solution of one problem gives rise to another problem one, but the process converges.

*Redundant information* The tape information of the simulated Turing machine will be stored in a redundant form, more precisely in the form of a block code.

*Redundant processing* The block code will be decoded, the retrieved information will be processed, and the result recorded. To carry out all this in a way that limits the propagation of faults, the tape will be split into tracks that can be handled separately, and the major processing steps will be carried out three times within one work period.

*Local repair* All the above process must be able to recover from a local burst of faults. For this, it is organized into a rigid, locally checkable structure with the help of local addresses, and some other tools like sweeps and short switchbacks (zigzags). The major tool of local correction, the local healing procedure, turns out to be the most complex part of the construction.

*Disturbed local repair* A careful organization of the healing procedure makes sure that even if a new burst interrupts it (or jumps into its middle), soon one or two new invocations of it will finish the job (whenever needed).

Here is some more detail. Each tape cell of the simulated machine  $M_2$  will be represented by a block of size  $Q$  of the simulating machine  $M_1$  called a *colony*. Each step of  $M_2$  will be simulated by a computation of  $M_1$  called a *work period*.

During this time, the head of  $M_1$  makes a number of sweeps over the current colony and its right neighbor colony, decodes the represented cell symbols, then computes and encodes the new symbols, and finally moves the head to the new position of the head of  $M_2$ .

In order to protect information from the propagation of errors, the tape of  $M_1$  is subdivided into *tracks*: each track corresponds to a *field* of a cell symbol of  $M_1$  viewed as a data record. Each stage of computation will be repeated three times. The results will be stored in separate tracks, and a final cell-by-cell majority vote will recover the result of the work period from them.

All this organization is controlled by a few key fields, for example a field called *Addr* showing the position of each cell in the colony, and a field *Sweep* showing the last sweep of the computation (along with its direction) that has been performed already. The technically most challenging part of the construction is the protection of this control information from bursts.

For example, a burst can reverse the head in the middle of a sweep. Our goal is that such structural disruptions be discovered locally, so we cannot allow the head to go far from the place where it was turned back. Therefore the head's movement will not be straight even during a single sweep: it will make frequent zigzags. This will trigger the healing procedure if for example a turn-back is detected.

It is a significant challenge that the healing procedure itself can be interrupted (or started) by a burst.

**Remark 2.1** This description uses some words in an informal way. Some of these will get precise definition later. For example, the word *colony* will have at least two formal definitions. From the point of view of the program (transition function), it is just a sequence of addresses from 0 to  $Q - 1$ . From the point of view of the analysis of the behavior of the machine, it is a sequence of actual adjacent tape cells with the address field having these values. ┘

## 2.2 Hierarchical construction

In order to build a machine resists faults occurring independently of each other with some small probability, we take the approach suggested in [5], and implemented in [3] and [4] for the case of one-dimensional cellular automata, with some ideas from the tiling application of [2]. We will build a *hierarchy of simulations*: machine  $M_1$  simulates machine  $M_2$  which simulates machine  $M_3$ , and so on. For simplicity we assume all these machines have the same program, and all simulations have the same block size.

One cell of machine  $M_3$  is simulated by one colony of machine  $M_2$ . Correspondingly, one cell of  $M_2$  is simulated by one colony of machine  $M_1$ . So one cell

of  $M_3$  is simulated by  $Q^2$  cells of  $M_1$ . Further, one step of machine  $M_3$  is simulated by one work period of  $M_2$  of, say,  $O(Q^2)$  steps. One step of  $M_2$  is simulated by one work period of  $M_1$ , so one step of  $M_3$  is simulated by  $O(Q^4)$  steps of  $M_1$ .

Per construction, machine  $M_1$  can withstand bursts of faults whose size is  $\leq \beta$  for some constant parameter  $\beta$ , that are separated by some  $O(Q^2)$  fault-free steps. Machines  $M_2, M_3, \dots$  have the same program, so it would be natural to expect that machine  $M_1$  can withstand also some *additional*, larger bursts of size  $\leq \beta Q$  if those are separated by at least  $O(Q^4)$  steps.

But a new obstacle arises. On the first level, damage caused by a big burst spans several colonies. The repair mechanism of machine  $M_1$  outlined in Section 2.1 above is too local to recover from such extensive damage. This cannot be allowed, since then the whole hierarchy would stop working. So we add a new mechanism to  $M_1$  that, more modestly, will just try to restore a large enough portion of the tape, so it can go on with the simulation of  $M_2$ , even if all original information was lost. For this,  $M_1$  may need to rewrite an area as large as a few colonies. This will enable the low-level healing procedure of machine  $M_2$  to restore eventually a higher-level order.

All machines above  $M_1$  in the hierarchy are “virtual”: the only hardware in the construction is machine  $M_1$ . Moreover, they will not be ordinary Turing machines, but *generalized* ones, with some new features that are not needed on the lowest level but seem necessary in a simulated Turing machine: for example they allow a positive distance between neighboring tape cells.

A tricky issue is “forced self-simulation”: while we are constructing machine  $M_1$  we want to give it the feature that it will simulate a machine  $M_2$  that works just like  $M_1$ . The “forced” feature means that this simulation should work without any written program (that could be corrupted).

This will be achieved by a construction similar to the proof of the Kleene’s fixed-point theorem (also called recursion theorem). We first fix a (simple) programming language to express the transition function of a Turing machine. We write an interpreter for it in this same language (just as compilers for the C language are sometimes written in C). The program of the transition function of  $M_2$  (essentially the same as that of  $M_1$ ) in this language, is a string that will be “hard-wired” into the transition function of  $M_1$ , so that  $M_1$ , at the start of each work period, can write it on a working track of the current colony. Then the work period will interpret it, applying it to the data found there, resulting in the simulation of  $M_2$ .

In this way, an infinite sequence of simulations arises, in order to withstand larger and larger but sparser and sparser bursts of faults.

Since the  $M_1$  uses the universal interpreter, which in turns simulates the same program, it is natural to ask how machine  $M_1$  simulates a given Turing machine

$G$  that does the actual useful computation? For this task, we set aside a separate track on each machine  $M_i$ , on which some arbitrary other Turing machine can be simulated. The higher the level of the machine  $M_k$  that performs this “side-simulation”, the higher the reliability. Thus, only the simulations  $M_k \rightarrow M_{k+1}$  are forced, without program (that is a hard-wired program): the side simulations can rely on written programs, since the firm structure in the hierarchy  $M_1, M_2, \dots$  will support them reliably.

### 2.3 From combinatorial to probabilistic noise

The construction we gave in the previous subsection was related to increasing bursts that are not frequent. In essence, that noise model is combinatorial. To deal with probabilistic noise combinatorially, we stratify the set of faulty times *Noise* as follows. For a series of parameters  $\beta_k, V_k$ , we first remove “isolated bursts” of type  $(\beta_1, V_1)$  of elements of this set. (The notion of “isolated bursts” of type  $(\beta, V)$  will be defined appropriately.) Then, we remove isolated bursts of type  $(\beta_2, V_2)$  from the remaining set, and so on. It will be shown that with the appropriate choice of parameters, with probability 1, eventually nothing is left over from the set *Noise*.

A composition of two reliable simulations is even more reliable. We will see that a sufficiently large hierarchy of such simulations resists probabilistic noise.

### 2.4 Difficulties

Let us spell out some of the main problems that the paper deals with, and some general ways they will be solved or avoided. Some more specific problems will be pointed out later, along with their solution.

*Non-aligned colonies* A large burst of  $M_1$  can modify the order of entire colonies or create new ones with gaps between them.

To overcome this problem conceptually, we introduce the notion of a *generalized Turing machine* allowing for non-adjacent cells. Each such machine has a parameter  $B$  called the *cell body size*. The cell body size of a Turing machine in Section 1.2 would still remain 1.

*No structure* What to do when the head is in a middle of an empty area where no structure exists? To ensure reliable passage across such areas, we will try to keep everything filled with cells, even if these are not part of the main computation.

*Clean areas* Noise can create areas over which the predictability of the simulated machine is limited. In these areas the (on this level) invisible structure of the underlying simulation may be destroyed. These areas should not simply be considered blank, since blankness implies predictable behavior. We call these areas



of “disordered”, and call the complement *clean*. Note that transition properties will allow making conclusions from cleanness, not from disorder. Examples below will show that when the head enters disorder, it can be “sucked in”.

*Extending cleanness* The definition of the generalized Turing machine stipulates a certain “magical” extension of clean intervals, and also the appearance of a clean “hole” around the head whenever it passes a certain amount of time in a small interval, and the cleaning of an interval if it is passed noiselessly a certain number of times. (Of course, these properties needs to be implemented in simulation, which is one of the main burdens of the actual construction.) Once an area is cleaned, it will be re-populated with new cells. Their content is not important, what matters is the restoration of predictability.

*Rebuilding* If local repair fails, a special rule will be invoked that reorganizes a larger part of the tape (of the size of a few colonies instead of only a few cells). This is the mechanism implementing the “magical” restoration on the next level.

The following examples show the difficulties that necessitate some of the complexities of the construction and proof.

**Example 2.2** (Need for zigging) When the head works in a colony of machine  $M_1$  performing a simulation of a cell of machine  $M_2$ , it works in sweeps across the colony. But a burst of faults could reverse the head in the middle of a sweep, leaving it uncompleted. This way a local burst could create non-local inconsistency.  $\lrcorner$

To handle the problem of Example 2.2, the head will proceed in zigzags: every step advancing the head in the simulation is followed by  $Z$  steps of going backward and forward again, just checking consistency (and starting a healing process if necessary). The parameter  $Z$  will be chosen appropriately.

**Example 2.3** (Need for feathering) Some big noise can create a number of intervals  $I_1, I_2, \dots, I_n$  consisting of colonies of machine  $M_1$ , each interval with its own simulated head, where the neighboring intervals are in no relation to each other. When the head is about to return from the end of  $I_k$  (never even to zig beyond it, see the discussion after Example 2.2), a burst can carry it over to  $I_{k+1}$  where the situation may be symmetric: it will continue the simulation that  $I_{k+1}$  is performing. (The rightmost colony of  $I_k$  and the leftmost colony of  $I_{k+1}$  need not be complete: what matters is only that the simulation in  $I_k$  would not bring the head beyond its right end, and the simulation in  $I_{k+1}$  would not bring the head beyond its left end.)

The head can be similarly captured to  $I_{k+2}$ , then much later back from  $I_{k+1}$  to  $I_k$ , and so on. This way the restoration of structure in  $M_2$  may be delayed too

long. ┘

The device by which we will mitigate the effect of this kind of capturing is another property of the the movement of the head which will call *feathering*: if the head turns back from a tape cell then next time it must go beyond. This requires a number of adjustments to the program (see later).

**Example 2.4** (Two slides over disorder) Consider two levels of simulation as outlined in Section 2.2: machine  $M_1$  simulates  $M_2$  which simulates  $M_3$ . The tape of  $M_1$  is subdivided into colonies of size  $Q_1$ . The tape content of the current colony of level 1 represents not only the content of the currently observed tape cell of machine  $M_2$ , but also its state. A burst on level 1 has size  $O(1)$ , while a burst on level 2 has size  $O(Q_1)$ .

Suppose that  $M_1$  is performing a simulation in colony  $C_0$ . An earlier big burst may have created a large interval  $D$  of disorder on the right of  $C_0$ , even reaching into  $C_0$ . For the moment, let  $C_0$  be called a *victim* colony. Assume that the left edge of  $D$  represents the last stage of a transfer operation to the right neighbor colony  $C_0 + Q_1$ . When the head, while performing its work in  $C_0$ , moves close to its right end, a small burst may carry it over into  $D$ . There it will be “captured”, and continue the (unintended) right transfer operation. This can carry the head, over several successful colony simulations in  $D$ , to some victim colony  $C_1$  on the right from which it will be captured to the right similarly. This can continue over new and new victim colonies  $C_i$  (with enough space between them to allow for new bursts to occur), all the way inside the disorder  $D$ . So the  $M_2$  cells in  $D$  will fail to simulate  $M_3$ .

After a while the head may return to the left in  $D$  (performing the simulations in its colonies). When it gets at the right end of a victim colony  $C_i$ , a burst might move it back there. There is a case when  $C_i$  now can just continue its simulation and then send the head further left: when before the head was captured on its right, it was in the last stage of simulating a left turn of the head of machine  $M_2$ .

In summary, a big burst can create a disordered area  $D$  which can capture the head and on which the head can slide forward and back without recreating any level of organization beyond the second one. ┘

**Example 2.5** (Unbounded number of slides over disorder) Let us describe a certain “organization” of a disordered area in which an unbounded number of passes may be required to restore order. For some  $n < 0$ , let the cells of  $M_1$  at positions  $x_{-Q_1}, \dots, x_n$ , where  $x_{i+1} = x_i + B_1$ , represent part of a healthy colony  $C(x_{-Q_1})$  starting at  $x_{-Q_1}$ , where  $x_n$  is the rightmost cell of  $C(x_{-Q_1})$  to which the head would come in the last sweep before the simulation will move to the *left* neighbor colony  $C(x_{-2Q_1})$ . Let them be followed by cells  $x_{n+1}, \dots, x_{Q_1-1}, \dots$  which represent the

last sweep of a transfer operation to the *right* neighbor colony  $C(x_0)$ . If the head is in cell  $x_n$ , a burst can transfer it to  $x_{n+1}$ . The cell state of  $M_2$  simulated by  $C(x_{-Q_1})$  need to be in *no relation* to the cell state of  $M_2$  simulated by  $C(x_0)$ . This was a capture of the head by a burst of  $M_1$  across the point 0, to the right.

We can repeat the capture scenario, say around points  $iQ_1Q_2$ , and this way cells of  $M_3$  simulated by  $M_2$  (simulated by  $M_1$ ) can be defined arbitrarily, with no consistency needed between any two neighbors. (We did not write  $iQ_1$  just in case bursts are not allowed in neighboring colonies.) In particular, we can define them to implement a *leftward* capture scenario via level 3 bursts at points  $iQ_1Q_2Q_3Q_4$ , allowing to simulate arbitrary cells of  $M_5$  with no consistency requirement between neighbors. So  $M_5$  could again implement a rightward capture scenario, and so on. In summary, a malicious arrangement of disorder and noise allows  $k$  passes after which the level of organization is still limited to level  $2k + 1$ .  $\lrcorner$

Our construction will ensure that, on the other hand,  $O(k)$  passes (free of  $k$ -level will restore organization to level  $k$ . This will property of the construction will be incorporated into our definition of a generalized Turing machine.

### 3 Notation

Most notational conventions given here are common; some other ones will also be useful.

*Natural numbers and integers* By  $\mathbb{Z}$  we denote the set of integers.

$$\begin{aligned}\mathbb{Z}_{>0} &= \{x : x \in \mathbb{Z}, x > 0\}, \\ \mathbb{Z}_{\geq 0} &= \mathbb{N} = \{x : x \in \mathbb{Z}, x \geq 0\}.\end{aligned}$$

*Intervals* We use the standard notation for intervals:

$$\begin{aligned}[a, b] &= \{x : a \leq x \leq b\}, & [a, b) &= \{x : a \leq x < b\}, \\ (a, b] &= \{x : a < x \leq b\}, & (a, b) &= \{x : a < x < b\}.\end{aligned}$$

We will also write  $[a, b)$  in place of  $[a, b) \cap \mathbb{Z}$ , whenever this leads to no confusion. Instead of  $[x + a, x + b)$ , sometimes we will write

$$x + [a, b).$$

*Ordered pairs* Ordered pairs are also denoted by  $(a, b)$ , but it will be clear from the context whether we are referring to an ordered pair or open interval.

*Comparing the order of a number and an interval* For a given number  $x$  and interval  $I$ , we write

$$x \geq I$$

if for every  $y \in I$ ,  $x \geq y$ .

*Distance* The distance between two real numbers  $x$  and  $y$  is defined in a usual way:

$$d(x, y) = |x - y|.$$

The *distance of a point  $x$  from interval  $I$*  is

$$d(x, I) = \min_{y \in I} d(x, y).$$

*Ball, neighborhood, ring, stripe* A ball of radius  $r > 0$ , centered at  $x$  is

$$B(x, r) = \{y : d(x, y) \leq r\}.$$

An  *$r$ -neighborhood of interval  $I$*  is

$$\{x : d(x, I) \leq r\}.$$

An  *$r$ -ring* around interval  $I$  is

$$\{x : d(x, I) \leq r \text{ and } x \notin I\}.$$

An  *$r$ -stripe to the right of interval  $I$*  is

$$\{x : d(x, I) \leq r \text{ and } x \notin I \text{ and } x > I\}.$$

*Logarithms* Unless specified differently, the base of logarithms throughout this work is 2.

## 4 Specifying a Turing machine

Let us introduce the tools allowing to describe the reliable Turing machine.

## 4.1 Universal Turing machine

We will describe our construction in terms of universal Turing machines, operating on binary strings as inputs and outputs. We define universal Turing machines in a way that allows for rather general “programs”.

**Definition 4.1** (Standard pairing) For a (possibly empty) binary string  $x = x(1) \cdots x(n)$  let us introduce the map

$$\langle x \rangle = 0^{|x|} 1x,$$

Now we encode pairs, triples, and so on, of binary strings as follows:

$$\begin{aligned}\langle s, t \rangle &= \langle s \rangle t, \\ \langle s, t, u \rangle &= \langle \langle s, t \rangle, u \rangle,\end{aligned}$$

and so on.

From now on, we will assume that our alphabets  $\Sigma, \Gamma$  are of the form  $\Sigma = \{0, 1\}^s, \Gamma = \{0, 1\}^g$ , that is our tape symbols and machine states are viewed as binary strings of a certain length. Also, if we write  $\langle i, u \rangle$  where  $i$  is some number, it is understood that the number  $i$  is represented in a standard way by a binary string.  $\lrcorner$

**Definition 4.2** (Computation result, universal machine) Assume that a Turing machine  $M$  starting on binary  $x$ , at some time  $t$  arrives at the first time at some final state. Then we look at the longest (possibly empty) binary string to be found starting at position 0 on the tape, and call it the *computation result*  $M(x)$ . We will write

$$M(x, y) = M(\langle x, y \rangle), \quad M(x, y, z) = M(\langle x, y, z \rangle),$$

and so on.

A Turing machine  $U$  is called *universal* among Turing machines with binary inputs and outputs, if for every Turing machine  $M$ , there is a binary string  $p_M$  such that for all  $x$  we have  $U(p_M, x) = M(x)$ . (This equality also means that the computation denoted on the left-hand side reaches a final state if and only if the computation on the right-hand side does.)  $\lrcorner$

Let us introduce a special kind of universal Turing machines, to be used in expressing the transition functions of other Turing machines. These are just the Turing machines for which the so-called  $s_{mn}$  theorem of recursion theory holds with  $s(x, y) = \langle x, y \rangle$ .

**Definition 4.3** (Flexible universal Turing machine) A universal Turing machine will be called *flexible* if whenever  $p$  has the form  $p = \langle p', p'' \rangle$  then

$$U(p, x) = U(p', \langle p'', x \rangle).$$

Even if  $x$  has the form  $x = \langle x', x'' \rangle$ , this definition chooses  $U(p', \langle p'', x \rangle)$  over  $U(\langle p, x' \rangle, x'')$ , that is starts with parsing the first argument (this process converges, since  $x$  is shorter than  $\langle x, y \rangle$ ).  $\lrcorner$

It is easy to see that there are flexible universal Turing machines. On input  $\langle p, x \rangle$ , a flexible machine first checks whether its “program”  $p$  has the form  $p = \langle p', p'' \rangle$ . If yes, then it applies  $p'$  to the pair  $\langle p'', x \rangle$ . (Otherwise it just applies  $p$  to  $x$ .)

**Definition 4.4** (Transition program) Consider an arbitrary Turing machine  $M$  with alphabet  $\Sigma$ , and transition function  $\tau$ . A binary string  $\pi$  will be called a *transition program* of  $M$  if whenever  $\tau(a) = (a', j)$  we have

$$U(\pi, a) = \langle a', j \rangle.$$

We will also require that the computation induced by the program makes  $O(|\pi| + |a|)$  left-right turns, over a length tape  $O(|\pi| + |a|)$ .  $\lrcorner$

The transition program just provides a way to compute the (local) transition function of  $M$  by the universal machine, it does not organize the rest of the simulation.

**Remark 4.5** In the construction of universal Turing machines provided by the textbooks (though not in the original one given by Turing), the program is generally a string encoding a table for the transition function  $\tau$  of the simulated machine  $M$ . Other types of program are imaginable: some simple transition functions can have much simpler programs. However, our fixed machine is good enough (similarly to the optimal machine for Kolmogorov complexity). If some machine  $U'$  simulates  $M$  via a very simple program  $q$ , then

$$M(x) = U'(q, x) = U(p_{U'}, \langle q, x \rangle) = U(\langle p_{U'}, q \rangle, x),$$

so  $U$  simulates this computation via the program  $\langle p_{U'}, q \rangle$ .  $\lrcorner$

## 4.2 Rule language

In what follows we will describe the generalized Turing machines  $M_k$  for all  $k$ . They are all similar, differing only in the parameter  $k$ ; the most important activity of  $M_k$  is to simulate  $M_{k+1}$ . The description will be uniform, except for the

parameter  $k$ . We will denote therefore  $M_k$  simply by  $M$ , and  $M_{k+1}$  by  $M^*$ . Similarly we will denote the block size  $Q_k$  of the block code of the simulation simply by  $Q$ .

Instead of writing a huge table describing the transition function  $\tau_k = \tau$ , we present the transition function as a set of *rules*. It will be then possible to write one *interpreter* program that carries out these rules; that program can be written for some fixed flexible universal machine Univ.

Each rule consists of some (nested) conditional statements, similar to the ones seen in an ordinary program: “*if condition then instruction else instruction*”, where the condition is testing values of some fields of the state and the observed cell, and the instruction can either be elementary, or itself a conditional statement. The elementary instructions are an *assignment* of a value to a field of the state or cell symbol, or a command to move the head. Rules can call other rules, but these calls will never form a cycle. Calling other rules is just a shorthand for nested conditions.

Even though rules are written like procedures of a program, they describe a single transition. When several consecutive statements are given, then they change different fields of the state or cell symbol, so they can be executed simultaneously.

Assignment of value  $x$  to a field  $y$  of the state or cell symbol will be denoted by  $y \leftarrow x$ . We will also use some conventions introduced by the C language: namely,  $x \leftarrow x + 1$  and  $x \leftarrow x - 1$  are abbreviated to  $x++$  and  $x--$  respectively.

Rules can also have parameters, like  $\text{Swing}(a, b, u, v)$ . Since each rule is called only a constant number of times in the whole program, the parametrized rule can be simply seen as a shorthand.

Mostly we will describe the rules using plain English, but it should always be clear that they are translatable into such rules.

For the machine  $M$  we are constructing, each tape symbol will be a tuple  $q = (q_1, q_2, \dots, q_k)$ , where the individual elements of the tuple will be called *fields*, and will have symbolic names. For example, we will have fields *Addr* and *Drift*, and may write  $q_1$  as  $q.\text{Addr}$  or just *Addr*,  $q_2$  as  $q.\text{Drift}$  or *Drift*, and so on.

In what follows we describe some of the most important fields we will use; others will be introduced later.

A properly formatted configuration of  $M$  splits the tape into blocks of  $Q$  consecutive cells called *colonies*. One colony of the tape of the simulating machine represents one cell of the simulated machine. The two colonies that correspond to the pair of cells that the simulated machine is scanning is called the *base colony-pair* (a precise definition will be based on the actual history of the work of  $M$ ), and its members will be called the *left* and *right base colony*. Sometimes the left base colony will just be called the *base colony*. Most of the computation proceeds

over the base colony-pair. The direction of the simulated head movement, once figured out by the computation, is called the *drift*. Two neighbor colonies may not be adjacent, in which case the cells will form a *bridge* between them. There will be a field whose value is called the *mode*:

$$Mode \in \{\text{Normal, Healing, Rebuilding}\}.$$

If for example its value in the current cell is Normal we will say that the computation is in *normal* mode. In this case, the machine is engaged in the regular business of simulation. The *healing* mode tries to correct some local fault due to a couple of neighboring bursts, while the *rebuilding* mode attempts to restore the colony structure on the scale of a couple of colonies.

The array of values of the same field of the cells will be called a *track*. Thus, we will talk about the *Hold* track of the tape, corresponding to the *Hold* field of cells.

Each field of a cell has also a possible value  $\emptyset$  whose approximate meaning is “undefined”.

Some fields and parameters are important enough to introduce them right away. The

*Info*

track of a colony of  $M$  contains the string that encodes the content of the simulated cell of  $M^*$ .

*Prog*

track stores the program of  $M^*$ , in an appropriate form to be interpreted by the simulation. The field

*Addr*

of the cell shows the position of the cell in its colony: it takes values in  $[0, Q)$ . The direction in  $\{-1, 1\}$  in which the simulated head moves will be denoted by

*Drift*.

The number of the last sweep of the work period will depend on the drift  $d$ , and will be denoted by

$$\text{Last}(d). \tag{4.1}$$



The

*Sweep*

field counts the sweeps that the head makes during the work period. In calculating parameters, we will make use of

$$V = \max(\text{Last}(-1), \text{Last}(1)). \quad (4.2)$$

Cells will be designated as belonging to a number of possible *kinds*, signaled by the field

*Kind*

with values

Member<sub>0</sub>, Member<sub>1</sub>, Bridge, Vac, Stem.

Here is a description of the role of these cell kinds. Cells of the base colony-pair are of type Member<sub>0</sub> and Member<sub>1</sub> respectively. If the base colony pair is not adjacent then there will be a *bridge* of  $< Q$  adjacent cells of type Bridge between them.

**Definition 4.6** We will say that the bridge *extends* a colony if its cells are adjacent to it. ┘

Under normal conditions, if the *Drift* track on the bridge has value 1 then it is extending the left base colony, otherwise is extending the right base colony. Cells of the bridge will continue the addressing (modulo  $Q$ ) of the colony it extends.

The stem kind is sometimes convenient when some cells need to be created temporarily that do not participate in any known colony structure. We will also try to keep all areas between colonies filled with (not necessarily adjacent) stem cells. For example the computation may find that a colony does not properly encode a tape cell by the required error-correcting code. Then we want to “kill” the whole colony. This will happen by turning the kind of each of its cells to Stem.

During healing, some special fields of the state and cell are used, they will be subfields of the field

$$\text{Heal}. \quad (4.3)$$

In particular, there will be a *Heal.Sweep* field.

Healing only changes the cells that need to be changed. But during rebuilding, the tape will also be used. we will work with subfields of the field *Rebuild*, and a cell will be called *marked for rebuilding* if *Rebuild.Sweep*  $\neq 0$ .

## 5 Exploiting structure in the noise

### 5.1 Sparsity

Let us introduce a technique connecting the combinatorial and probabilistic noise models.

**Definition 5.1** (Centered rectangles, isolation) Let  $\mathbf{r} = (r_1, r_2)$ ,  $r_1, r_2 \geq 0$ , be a two-dimensional nonnegative vector. An *rectangle* of radius  $\mathbf{r}$  centered at  $\mathbf{x}$  is

$$B(\mathbf{x}, \mathbf{r}) = \{\mathbf{y} : |y_i - x_i| \leq r_i, i = 1, 2\}. \quad (5.1)$$

Let  $E \subseteq \mathbb{Z}^2$  be a two-dimensional set. A point  $\mathbf{x}$  of  $E$  is  $(\mathbf{r}, \mathbf{r}^*)$ -isolated if

$$E \cap B(\mathbf{x}, \mathbf{r}^*) \subseteq B(\mathbf{x}, \mathbf{r}).$$

Set  $E$  is  $(\mathbf{r}, \mathbf{r}^*)$ -sparse if  $D(E, \mathbf{r}, \mathbf{r}^*) = \emptyset$ , that is it consists of  $(\mathbf{r}, \mathbf{r}^*)$ -isolated points. Let

$$D(E, \mathbf{r}, \mathbf{r}^*) = \{\mathbf{x} \in E : \mathbf{x} \text{ is not } (\mathbf{r}, \mathbf{r}^*)\text{-isolated from } E\}. \quad (5.2)$$

⌋

**Definition 5.2** (Sparsity) Let

$$\beta \geq 9 \quad (5.3)$$

be a parameter, and let

$$0 < B_1 < B_2 < \cdots, \quad T_1 < T_2 < \cdots, \quad 1 \leq \gamma_1 \leq \gamma_2 \leq \cdots$$

be sequences of positive integers to be fixed later. For a two-dimensional set  $E$ , let  $E^{(1)} = E$ . For  $k > 1$  we define recursively:

$$E^{(k+1)} = D(E^{(k)}, \beta(B_k, T_k), \gamma_k(B_{k+1}, T_{k+1})). \quad (5.4)$$

Set  $E^{(k)}$  is called the  $k$ -th residue of  $E$ . It is  $k$ -sparse if  $E^{(k+1)} = \emptyset$ . It is simply sparse if  $\bigcap_k E^{(k)} = \emptyset$ .

When  $E = E^{(k)}$  and  $k$  is known then we will denote  $E^{(k+1)}$  simply by  $E^*$ . ⌋

The following lemma connects the above defined sparsity notions to the requirement of small fault probability. It is formulated somewhat redundantly, for easier application.

**Lemma 5.3** (Sparsity) *Let  $Q_k = B_{k+1}/B_k$ ,  $U_k = T_{k+1}/T_k$ , and*

$$\lim_{k \rightarrow \infty} \frac{\log(\gamma_k U_k Q_k)}{1.5^k} = 0. \quad (5.5)$$

*For sufficiently small  $\varepsilon$ , for every  $k \geq 1$  the following holds. Let  $E \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0}$  be a random set with the property that each pair  $(p, t)$  belongs to  $E$  independently from the other ones with probability  $\leq \varepsilon$ .*

*Then for each point  $\mathbf{x}$  and each  $k$ ,*

$$\mathbb{P}\{B(\mathbf{x}, (B_k, T_k)) \cap E^{(k)} \neq \emptyset\} < 2\varepsilon \cdot 2^{-1.5^k}.$$

*As a consequence, the set  $E$  is sparse with probability 1.*

*Proof.* Let  $k = 1$ . Rectangle  $B(\mathbf{x}, (B_1, T_1))$  is a single point, hence the probability of our event is  $< \varepsilon$ . Let us prove the inequality by induction, for  $k + 1$ .

Note that our event depends at most on the rectangle  $B(\mathbf{x}, 3(B_k, T_k))$ . Let

$$p_k = 2\varepsilon \cdot 2^{-1.5^k}.$$

Suppose  $\mathbf{y} \in E^{(k)} \cap B(\mathbf{x}, \gamma_k(B_{k+1}, T_{k+1}))$ . Then, according to the definition of  $E^{(k)}$ , there is a point

$$\mathbf{z} \in B(\mathbf{y}, \gamma_k(B_{k+1}, T_{k+1})) \cap E^{(k)} \setminus B(\mathbf{y}, \beta(B_k, T_k)). \quad (5.6)$$

Consider a standard partition of the (two-dimensional) space-time into rectangles  $K_p = \mathbf{c}_p + [-B_k, B_k] \times [-T_k, T_k]$  with centers  $\mathbf{c}_1, \mathbf{c}_2, \dots$ . The rectangles  $K_i, K_j$  containing  $\mathbf{y}$  and  $\mathbf{z}$  respectively intersect  $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$ . The triple-size rectangles  $K'_i = \mathbf{c}_i + [-3B_k, 3B_k] \times [-3T_k, 3T_k]$  and  $K'_j$  are disjoint, since (5.3) and (5.6) imply  $|\mathbf{y}_1 - \mathbf{z}_1| > \beta B_k$  and  $|\mathbf{y}_2 - \mathbf{z}_2| > \beta T_k$ .

The set  $E^{(k)}$  must intersect two rectangles  $K_i, K_j$  of size  $2(B_k, T_k)$  separated by at least  $4(B_k, T_k)$ , of the big rectangle  $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$ .

By the inductive hypothesis, the event  $\mathcal{F}_i$  that  $K_i$  intersects  $E_k$  has probability bound  $p_k$ . It is independent of the event  $\mathcal{F}_j$ , since these events depend only on the triple size disjoint rectangles  $K'_i$  and  $K'_j$ .

The probability that both of these events hold is at most  $p_k^2$ . The number of possible rectangles  $K_p$  intersecting  $B(\mathbf{x}, 2\gamma_k(B_{k+1}, T_{k+1}))$  is at most  $C_k := ((2\gamma_k^2 U_k Q_k) + 2)^2$ , so the number of possible pairs of rectangles is at most  $C_k^2/2$ , bounding the probability of our event by

$$\begin{aligned} C_k^2 p_k^2 / 2 &= 2C_k^2 \varepsilon^2 2^{-1.5^{k+1}} \cdot 2^{-0.5 \cdot 1.5^k} \\ &= 2\varepsilon 2^{-1.5^{k+1}} \cdot \varepsilon C_k^2 2^{-0.5 \cdot 1.5^k}. \end{aligned}$$

Since  $\lim_k \frac{\log(\gamma_k U_k Q_k)}{1.5^k} = 0$ , the last factor is  $\leq 1$  for sufficiently small  $\varepsilon$ .  $\square$

In our construction we will choose

$$\gamma_k = \Omega(k^2), \quad Q_k = \Omega(\gamma_k^2), \quad U_k = \Omega(Q_k^2),$$

while still satisfying (5.5).

## 5.2 Error-correcting code

Let us add error-correcting features to block codes introduced in Definition 1.2.

**Definition 5.4** (Error-correcting code) A block code is  $(\beta, t)$ -burst-error-correcting, if for all  $x \in \Sigma_2$ ,  $y \in \Sigma_1^Q$  we have  $\psi^*(y) = x$  whenever  $y$  differs from  $\psi_*(x)$  in at most  $t$  intervals of size  $\leq \beta$ .

For such a code, we will call a word  $y \in \Sigma_1^Q$  is  $r$ -compliant if it differs from a codeword of the code by at most  $r$  intervals of size  $\leq \beta$ .  $\lrcorner$

**Example 5.5** (Repetition code) Suppose that  $Q \geq 3\beta$  is divisible by 3,  $\Sigma_2 = \Sigma_1^{Q/3}$ ,  $\psi_*(x) = xxx$ . Let  $\psi^*(y)$  be obtained as follows. If  $y = y(1) \dots y(Q)$ , then  $x = \psi^*(y)$  is defined as follows:  $x(i) = \text{maj}(y(i), y(i + Q/3), y(i + 2Q/3))$ . For all  $\beta \leq Q/3$ , this is a  $(\beta, 1)$ -burst-error-correcting code.

If we repeat 5 times instead of 3, we get a  $(\beta, 2)$ -burst-error-correcting code. Let us note that there are much more efficient such codes than just repetition.  $\lrcorner$

Consider a Turing machine  $(\Gamma, \Sigma, \tau, q_{\text{start}}, F)$  (actually a generalized one to be defined later) simulating some Turing machine  $(\Gamma^*, \Sigma^*, \tau^*, q_{\text{start}}^*, F^*)$ . We will assume that  $\Gamma^* \cup \{\emptyset\}$ , and the alphabet  $\Sigma^*$  are subsets of the set of binary strings  $\{0, 1\}^l$  for some  $l < Q$  (we can always ignore some states or tape symbols, if we want).

**Definition 5.6** (Interior) Let

$$\text{PadLen}$$

be a parameter to be defined later (in (7.4)). Consider an interval  $I$  of neighbor cells. A cell belongs to the *interior* of  $I$  if there are at least  $\text{PadLen}$  neighbors between it and the complement of  $I$ . In particular, we will talk about the interior of a colony.  $\lrcorner$

We will store the coded information in the interior of the colony, since it is more exposed to errors near the boundaries. So let  $(v_*, v^*)$  be a  $(\beta, 2)$ -burst-error-correcting block code

$$v_* : \{0, 1\}^l \cup \{\emptyset\} \rightarrow \{0, 1\}^{(Q-2 \cdot \text{PadLen})B}.$$

We could use, for example, the repetition code of Example 5.5. Other codes are also appropriate, but we require that they have some fixed programs  $p_{\text{encode}}$ ,  $p_{\text{decode}}$  on the universal machine Univ, in the following sense:

$$\nu_*(x) = \text{Univ}(p_{\text{encode}}, x), \quad \nu^*(y) = \text{Univ}(p_{\text{decode}}, y).$$

Also, these programs must work in quadratic time and linear space on a one-tape Turing machine (as the repetition code certainly does).

Let us now define the block code  $(\psi_*, \psi^*)$  used in the definition of the configuration code  $(\varphi_*, \varphi^*)$  as outlined in Section 6.3. We define

$$\psi_*(a) = 0^{\text{PadLen}} \nu_*(a) 0^{\text{PadLen}}. \quad (5.7)$$

It will be easy to compute the configuration code from  $\psi_*$ , once we know what fields there are which ones need initialization.

The decoded value  $\psi^*(x)$  is obtained by first removing  $\text{PadLen}$  symbols from both ends of  $x$  to get  $x'$ , and then computing  $\nu^*(x')$ .

## 6 The model

Recall the definition of sparsity in Section 5.1: there will be a sequence  $0 < B_1 < B_2 < \dots$  of “scales” in space and a sequence  $T_1 < T_2 < \dots$  of scales in time, and a constant  $\beta$ . We will define a sequence of simulations  $M_1 \rightarrow M_2 \rightarrow \dots$  where each  $M_k$  is a machine simulating one on a higher level. For simplicity, we will use the notation  $M = M_k$ ,  $M^* = M_{k+1}$ , and similarly for the other parameters, for example  $B, T, Q, U$ . As already indicated in Section 2.2, these machines will generalize ordinary Turing machines, with a number of new features.

**Notation 6.1** For an interval  $I = [a, b)$  and some  $c \geq 0$  let

$$\text{Int}(I, c) = [a + c, b - c).$$

Similarly for intervals  $(a, b]$ ,  $[a, b]$ ,  $(a, b)$ . ┘

### 6.1 Generalized Turing machine

Standard Turing machines do not have operations like “creation” or “killing” of cells, nor do they allow for cells to be non-adjacent. We introduce here a *generalized Turing machine*. It depends on an integer  $B \geq 1$  that denotes the cell body size, and an upper bound  $T$  on the transition time. These parameters are convenient since they provide the illusion that the different Turing machines in the hierarchy of simulations all operate on the same linear space and the same time interval.

Even if the notions of cells, alphabet and state are different for each machine of the hierarchy, at least the notion of a *location on the tape* is the same. There will also be a *pass number* parameter  $\pi$ , whose meaning will be explained in the definition of trajectories.

**Definition 6.2** (Generalized Turing machine) A *generalized Turing machine*  $M$  is defined by a tuple

$$(\Sigma, \tau, Adj, B, T, \pi), \quad (6.1)$$

where  $\Sigma$  is a finite set called the *alphabet*,

$$\tau : \Sigma^2 \rightarrow \Sigma^2 \times \{-1, 1\}$$

is the *transition function*,  $Adj$  is a function on the alphabet (can be called a “field” if a symbol is viewed as a piece of data, a record with several fields).  $Adj : \Sigma \rightarrow \{0, 1\}$  will show whether the last move was from an adjacent cell. The integer  $B \geq 1$  is called the *cell body size*, and the real number  $T$  is a bound on the transition time. The integer  $\pi$  will be the number of passes needed to clean an area (see below). Among the elements of the tape alphabet  $\Sigma$ , we distinguish the elements 0, 1, Bad, Vac. The role of the symbols Bad and Vac will be clarified below.  $\square$

The definition of a configuration below introduces a new concept, the *current cell-pair* position  $\hat{\mathbf{h}} = (\hat{h}_0, \hat{h}_1)$ . (We will continue calling  $\hat{h}_0$  the *current cell*.) Here is some explanation for the need of this new variable. A generalized Turing machine  $M_2$  may be simulated by some lower-level (generalized) Turing machine  $M_1$ . Not all details of the tape content of  $M_1$  are visible on the level of  $M_2$ : what is visible is the content of the cells of  $M_2$  decoded from those of  $M_1$ . The currently observed cell-pair in the simulated machine  $M_2$  is  $\hat{\mathbf{h}}$ . On the other hand the true head position of  $M_1$  is  $h$  which may perform a lot of oscillatory movement not directly relevant to the work of  $M_2$ .

**Definition 6.3** (Configuration) Consider a generalized Turing machine (6.1). A *configuration* is a tuple

$$(A, h, \hat{\mathbf{h}}),$$

where  $A : \mathbb{Z} \rightarrow \Sigma$ ,  $h, \hat{h}_j \in \mathbb{Z}$ . The array  $A$  is the tape configuration. As in (1.1) before,  $h$  is the head position, but it may differ from the current cell  $\hat{h}_0$ , so now if  $\xi = (A, h, \hat{\mathbf{h}})$  then we write

$$\xi.\text{tape} = A, \quad \xi.\text{pos} = h, \quad \xi.\text{cur-cell}_j = \hat{h}_j. \quad (6.2)$$

A point  $p$  is *clean* if  $A(p) \neq \text{Bad}$ . A set of points is *clean* if it consists of clean points.

We say that there is a *cell* at a position  $p \in \mathbb{Z}$  if the interval  $p + [0, B)$  is clean and  $A(p) \neq \text{Vac}$ . In this case, we call the interval  $p + [0, B)$  the *body* of this cell. Cells must be at distance  $\geq B$  from each other, that is their bodies must not intersect. If they are at a distance  $< 2B$  from each other then they are called *neighbors*. They are called *adjacent* if the distance is exactly  $B$ . A configuration will always have the following property:

C1) Any endpoint of a maximal clean interval is the end of a cell body in that direction.

Whenever the interval  $h + [4B, 4B)$  is clean there must be a pair of neighbor cells at some positions  $\hat{h}_0, \hat{h}_1$  within this interval called the *current cell-pair*.

The array  $A$  is Vac everywhere but in finitely many positions. Let

$$\text{Configs}_M$$

denote the set of all possible configurations of a Turing machine  $M$ . ┘

All the above definitions can clearly be localized to define a configuration *over a space interval*  $I$ , where it is always understood that  $h \in I$ , that is  $I$  contains the head.

**Definition 6.4** (Local configuration, replacement) A *local configuration* on a (finite or infinite) interval  $I$  is given by values assigned to the cells of  $I$ , along with the following information: whether the head is to the left of, to the right of or inside  $I$ , and if it is inside, on which cell.

If  $I'$  is a subinterval of  $I$ , then a local configuration  $\xi$  on  $I$  clearly gives rise to a local configuration  $\xi(I')$  on  $I'$  as well, called its *subconfiguration*: If the head of  $\xi$  was in  $I$  and it was for example to the left of  $I'$ , then now  $\xi(I')$  just says that it is to the left, without specifying its position.

Let  $\xi$  be a configuration and  $\zeta(I)$  a local configuration that contains the head if and only if  $\xi(I)$  contains the head. Then the configuration  $\xi|\zeta(I)$  is obtained by replacing  $\xi$  with  $\zeta$  over the interval  $I$ , further if  $\xi$  contains the head then also replacing  $\xi.\text{pos}$  with  $\zeta.\text{pos}$ . ┘

It is natural to name a sequence of configurations that is conceivable as a computation (faulty or not) of a Turing machine as “history”. The histories that obey the transition function then could be called “trajectories”. In what follows we will stretch this notion to encompass also some limited violations of the transition function. In connection with any underlying Turing machine with a given starting configuration, we will denote by

$$\text{Noise} \subseteq \mathbb{Z} \times \mathbb{Z}_{\geq 0} \tag{6.3}$$

the set of space-time points  $(p, t)$ , such that a fault occurs at time  $t$  when the head is at position  $p$ .

**Definition 6.5** (History) For a generalized Turing machine (6.1), consider a sequence  $\eta = (\eta(0), \eta(1), \dots)$  of configurations with  $\eta(t) = (A(\cdot, t), h(t), \hat{\mathbf{h}}(t))$ , along with a noise set *Noise*. The *switching times* of this sequence are the times  $t$  when the position or content of the current cell-pair changes. The interval between two consecutive switching times is the *dwell period*. The pair

$$(\eta, \text{Noise})$$

will be called a *history* of machine  $M$  if the following conditions hold.

- $|h(t) - h(t')| \leq |t' - t|$ .
- In two consecutive configurations, content  $A(p, t)$  of the positions  $p$  not in  $h(t) + [-2B, 2B]$ , remains the same: for example  $A(n, t + 1) = A(n, t)$  for all  $n \notin h(t) + [-2B, 2B]$ .
- At each noise-free switching time the head is on the new current cell:  $\hat{h}_0(t) = h(t)$ . In particular, when at a switching time a current cell becomes Vac, the head must already be on another (current) cell.
- The length of any noise-free dwell period in which the head is staying on clean positions is at most  $T$ .

Let

$$\text{Histories}_M$$

denote the set of all possible histories of  $M$ .

We say that a cell *dies* in a history if it becomes Vac.

All the above definition can be *localized* to define a history *over a space-time rectangle*  $I \times J$ , where it is always understood that  $h \in I$  for all times  $t \in J$ , that is  $I$  contains the head throughout the time interval considered.  $\lrcorner$

In Section 6.4 below, we will define a certain subset of possible histories, called *trajectories*. The set of trajectories of  $M$  will be denoted by

$$\text{Trajectories}_M.$$

## 6.2 Simulation

Until this moment, we used the term “simulation” informally, to denote a correspondence between configurations of two machines which remains preserved during the computation. In the formal definition, this correspondence will essentially be a code  $\varphi = (\varphi_*, \varphi^*)$ . The *decoding* part of the code is the more important. We want to say that machine  $M_1$  simulates machine  $M_2$  via simulation  $\varphi$  if whenever



$(\eta, \text{Noise})$  is a trajectory of  $M_1$  then  $(\eta^*, \text{Noise}^*)$ , defined by  $\eta^*(\cdot, t) = \varphi^*(\eta(\cdot, t))$ , is a trajectory of  $M_1$ . Here,  $\text{Noise}^*$  is computed by an appropriate mapping.

We will make, however, two refinements. First, we may weaken the condition by requiring this only for those  $\eta$  for which the initial configuration  $\eta(\cdot, 0)$  has been obtained by encoding, that is it has the form  $\eta(\cdot, 0) = \varphi_*(\xi)$ . The encoding function gets a role this way in the definition, after all.

But there is a more complex refinement. When a colony is in transition between encoding one simulated value to encoding another one, there may be times when the value represented by it before the transition is already not decodable from it, and the value after the transition is not yet decodable from it. So we will define simulation decoding as a mapping  $\Phi^*$  between *histories*, not just configurations. This allows a certain amount of “looking back”: the map  $\Phi^*$  can depend on the configurations at the beginning of the “work period”.

It is the mapping  $\Phi^*$  that will also define  $\text{Noise}^*$ . A history was defined above in Definition 6.5 as a pair  $(\eta, \text{Noise})$ , so we will have  $\Phi^*(\eta, \text{Noise}) = (\eta^*, \text{Noise}^*)$ . The meaning of  $\text{Noise}^*$  will be, just as in Definition 5.2 of sparsity: it will be obtained from  $\text{Noise}$  by deleting those small isolated parts that the error-correcting simulation can deal with.

**Definition 6.6** (Simulation) Let  $M_1, M_2$  be two generalized Turing machines, and let

$$\varphi_* : \text{Configs}_{M_2} \rightarrow \text{Configs}_{M_1}$$

be a mapping from configurations of  $M_2$  to those of  $M_1$ , such that it maps starting configurations into starting configurations. We will call such a map a *configuration encoding*. Let

$$\Phi^* : \text{Histories}_{M_1} \rightarrow \text{Histories}_{M_2}$$

be a mapping. The pair  $(\varphi_*, \Phi^*)$  is called a *simulation* (of  $M_2$  by  $M_1$ ) if for every trajectory  $(\eta, \text{Noise})$  with initial configuration  $\eta(\cdot, 0) = \varphi_*(\xi)$ , the history  $(\eta^*, \text{Noise}^*) = \Phi^*(\eta, \text{Noise})$  is a trajectory of machine  $M_2$ .

We say that  $M_1$  *simulates*  $M_2$  if there is a simulation  $(\varphi_*, \Phi^*)$  of  $M_2$  by  $M_1$ .  $\lrcorner$

### 6.3 Hierarchical codes

Recall the notion of a code in Definition 1.2.

**Definition 6.7** (Code on configurations) Consider two generalized Turing machines  $M_1, M_2$  with the corresponding alphabets and transition functions, and an integer  $Q \geq 1$ . We require

$$B_2 = QB_1. \tag{6.4}$$

Assume that a block code  $\psi_* : \Sigma_2 \rightarrow \Sigma_1^Q$  is given, with an appropriate decoding function,  $\psi^*$ . Symbol  $a \in \Sigma_2$ , is interpreted the content of some tape square.

This block code gives rise to a *code on configurations*, that is a pair of functions

$$\varphi_* : \text{Confs}_{M_2} \rightarrow \text{Confs}_{M_1}, \quad \varphi^* : \text{Confs}_{M_1} \rightarrow \text{Confs}_{M_2}$$

that encodes some (initial) configurations  $\xi$  of  $M_2$  into configurations of  $M_1$ . Let  $\xi$  be a configuration of  $M_2$  with  $\xi.\text{cur-cell}_j = \xi.\text{pos} + jB_2$ . We set  $\varphi_*(\xi).\text{pos} = \xi.\text{pos}$ ,  $\varphi_*(\xi).\text{cur-cell}_j = \varphi_*(\xi).\text{pos} + jB_1$ , and

$$\varphi_*(\xi).\text{tape}(iB_2, \dots, (i+1)B_2 - B_1) = \psi_*(\xi.\text{tape}(i)).$$

A configuration  $\xi$  is called a *code configuration* if it has the form  $\xi = \varphi_*(\zeta)$ .  $\lrcorner$

**Definition 6.8** (Hierarchical code) For  $k \geq 1$ , let  $\Sigma_k$  be an alphabet, of a generalized Turing machine  $M_k$ . Let  $Q_k > 0$  be an integer colony size, let  $\varphi_k$  be a code on configurations defined by a block code

$$\psi_k : \Sigma_{k+1} \rightarrow \Sigma_k^{Q_k}$$

as in Definition 6.7. The sequence  $(\Sigma_k, \varphi_k)$ ,  $(k \geq 1)$ , is called a *hierarchical code*. For the given hierarchical code, the configuration  $\xi^1$  of  $M_1$  is called a *hierarchical code configuration* if a sequence of configurations  $\xi^2, \xi^3, \dots$  of  $M_2, M_3, \dots$  exists with

$$\xi^k = \varphi_{*k}(\xi^{k+1})$$

for all  $k$ . (Of course, then whole sequence is determined by  $\xi^1$ .)

Let  $M_1, M_2, \dots$  be a sequence of generalized Turing machines, let  $\varphi_1, \varphi_2, \dots$  be a hierarchical code for this sequence, let  $\xi^1$  be a hierarchical code configuration for it, where  $\xi^k$  is an initial configuration of  $M_k$  for each  $k$ . Let further be a sequence of mappings  $\Phi_1^*, \Phi_2^*, \dots$  be given such that for each  $k$ , the pair  $(\varphi_{k*}, \Phi_k^*)$ , is a simulation of  $M_{k+1}$  by  $M_k$ . Such an object is called a *tower*.  $\lrcorner$

The main task of the work will be the definition of a tower, since the simulation property is highly nontrivial.

## 6.4 Trajectories

This subsection completes the definition of a generalized Turing machine. The set of trajectories  $\text{Trajectories}_M$  is defined in terms of constraints imposed on the burst-free parts of a history. We discuss these properties first informally.

*Transition Function* This property says (in more precise terms) that in the absence of noise and in a clean area, the transition function is obeyed.

*The Spill Bound* limits the extent to which a disordered interval can spread while the head is in it.

*Escape* limits the time that the head can spend in a small area.

*Attack Cleaning* helps erode the disorder as the head repeatedly enters and leaves it.

*Pass Cleaning* limits the number of back-and-forth slides that can happen over a large disordered area (under certain conditions). Otherwise every time the head reaches, say, its left end, a new burst could expand the disorder. As these bursts are not visible in the simulated trajectory  $\eta^*$ , the Spill Bound property of  $\eta^*$  could be violated.

**Definition 6.9** Suppose that at times  $t'$  before a switching time  $t$  but after any previous switch, the current cell-pair has state  $\mathbf{a}$ , further after the switch the same cell pair has state  $\mathbf{a}'$  (including when one of these cells dies).

We say that the switch is *dictated by the transition function* if

$$(\mathbf{a}', \text{sign}(\hat{h}_0(t') - \hat{h}_0(t))) = \tau(\mathbf{a}),$$

further  $a'_0 \text{Adj} = 1$  if and only if  $\hat{h}_0(t') - \hat{h}_0(t) \in \{-B, 0, B\}$ . ┘

We will need the following constants, to be fixed later.

$$c_{\text{attack}} < \beta/2, c_{\text{spill}}, \theta, c_{\text{pass}} \leq \theta/6, c_{\text{esc}}. \quad (6.5)$$

**Peter remark 1.** Some details of the following definition need to be adjusted yet. ┘

**Definition 6.10** (Trajectory) A history  $(\eta, \text{Noise})$  of a generalized Turing machine (6.1) with  $\eta(t) = (\mathcal{A}(t), h(t), \hat{\mathbf{h}}(t))$  is called a *trajectory* of  $M$  if the following conditions hold, in any noise-free time interval  $J$ .

*Transition Function* Suppose that there is a switch, and the current cell-pair is inside the clean area, by a distance of at least  $2.5B$ . Then the new state of the current cell-pair, and the direction towards the new current head position are dictated by the transition function. If a new current cell did not exist before then it is adjacent to one of the previous ones before the switch. The only change on the tape occurs on the interval enclosing the new and old current cells. Further, the length of the dwell period is bounded by  $T$ .

*Spill Bound* An interval of disorder can spread by at most  $c_{\text{spill}}B$  on either side while it contains the head.

*Escape* The head will leave any interval of size  $\leq \lambda B$  with  $1 \leq \lambda \leq 3\beta$  within time  $c_{\text{esc}}\lambda^2 T$ .

*Attack cleaning* Suppose that the current cell-pair is at the end of a maximal clean interval of size  $\geq (c_{\text{attack}} + 1)B$ . Suppose further that the transition function directs the head right. Then by the time the head comes back to  $x - c_{\text{attack}}B$ , the right end of the clean interval containing it advances to the right by at least  $B$ . A similar property is required when “left” and “right” are interchanged.

*Pass Cleaning* This property will be defined below in Definition 6.14.

┘

To motivate the Pass Cleaning property, the following example shows that in our noise model, the number of passes over some disorder needed to eliminate it cannot be a universal constant: it has to depend on the machine.

**Example 6.11** (Many slides over disorder) Let us describe a certain “organization” of a disordered area along with a scenario in which an unbounded number of passes may be required to restore order. For some  $n < 0$ , let the cells of  $M_1$  at positions  $x_{-Q_1}, \dots, x_n$ , where  $x_{i+1} = x_i + B_1$ , represent part of a healthy colony  $C(x_{-Q_1})$  starting at  $x_{-Q_1}$ , where  $x_n$  is the rightmost cell of  $C(x_{-Q_1})$  to which the head would come in the last sweep before the simulation will move to the *left* neighbor colony  $C(x_{-2Q_1})$ . Let them be followed by cells  $x_{n+1}, \dots, x_{Q_1-1}, \dots$  which represent the last sweep of a transfer operation to the *right* neighbor colony  $C(x_0)$ . When the head is in cell  $x_n$ , a burst transfers it to  $x_{n+1}$ . The cell state of  $M_2$  simulated by  $C(x_{-Q_1})$  needs to be in *no relation* to the cell state of  $M_2$  simulated by  $C(x_0)$ . This was a capture of the head by a burst of  $M_1$  across the point 0, to the right.

We can repeat the capture scenario, say around points  $iQ_1Q_2$ , and this way cells of  $M_3$  simulated by  $M_2$  (simulated by  $M_1$ ) can be defined arbitrarily, with no consistency needed between any two neighbors. (We did not write  $iQ_1$  just in case bursts are not allowed in neighboring colonies.) In particular, we can define them to implement a *leftward* capture scenario via level 3 bursts at points  $iQ_1Q_2Q_3Q_4$ , allowing to simulate arbitrary cells of  $M_5$  with no consistency requirement between neighbors. So  $M_5$  could again implement a *rightward* capture scenario, and so on. In summary, a malicious arrangement of disorder and noise allows  $k$  passes after which the level of organization is still limited to level  $2k + 1$ . ┘

Our construction will set  $\pi = O(k)$  for a machine on level  $k$  of the simulation, that is  $O(k)$  passes will restore organization to level  $k$ . (Essentially, the third slide should raise the level.) But some restriction is needed: if between the noiseless passes there is a large number of *intrusions* that are not passes, then each of these intrusions can bring a little new disorder, possibly undoing the effect of passes. Just limiting the *number* of intrusions is not sufficient, since there can be too many local intrusions. In our noise model, a space-time area is noiseless for the simulated

machine  $M^*$  if, essentially, every  $2\gamma \cdot (QB) \times (UT)$  rectangle contains at most one burst (of size  $\beta \cdot B \times T$ ). So instead of limiting the number of intrusions, we will limit the number of a certain kind of rectangles covering them.

**Definition 6.12** For a space-time path  $P$  of the head and an interval  $I$ , let  $P_{u:v}$  be its part between  $u$  and  $v$ . We write  $P_{:v}$  for the part from start to  $v$  and  $P_u$  for the part from  $u$  to end.

A part  $P_{u:v}$  is a *pass* of  $I$  if  $P$  enters  $I$  at time  $u$  and leaves it on the other side at time  $v$  (without leaving  $I$  before). We say that  $P$  *intrudes*  $I$  if it enters and leaves  $I$  on the same side.  $\lrcorner$

**Definition 6.13** ( $W$ -step length) Consider a path  $P$  over some time interval  $[t, u]$ . Break it up into segments of the form  $[a, b]$  in such a way that each segment has size  $\leq W$  and only the last segment can have size  $< W$ . Such a segmentation is called a  $W$ -segmentation of  $P$ , and the number of segments is the  $W$ -count of  $P$ .

If a path  $P$  may pass or intrude some interval  $I$  several times then we call the  $W$ -count of intrusions of  $P$  into  $I$  the sum of the  $W$ -counts of all intrusions.  $\lrcorner$

Note that if  $P$  is noiseless and  $W = \lambda B$  with  $1 \leq \lambda \leq 3\beta$  then by the Escape property, the time spent in each segment of a  $W$ -segmentation of  $W$  is at most  $c_{\text{esc}}\lambda^2 T$ .

Recall the constants defined in (6.5).

**Definition 6.14** (Pass cleaning property) Suppose that a path  $P$  passes interval  $I$  at least  $\pi$  times, and the  $\theta B$ -count of the intrusions of  $P$  into  $I$  is at most  $\pi^2$ . Then during  $P$  the interior  $\text{Int}(I, c_{\text{pass}}B)$  of the interval  $I$  becomes clean.  $\lrcorner$

The trajectory properties can clearly be localized to some space-time rectangle just as the definition of history was.

## 7 Simulation structure

In what follows we will describe the program of the reliable Turing machine (more precisely, a simulation of each  $M_{k+1}$  by  $M_k$  as defined above). Most of the time, we will just refer to  $M_k$  as  $M$  and to  $M_{k+1}$  as  $M^*$ . Cells will be grouped into colonies, where  $Q = B^*/B$  is the colony size. The behavior of the head on each colony simulates the head of  $M^*$  on the corresponding cell of  $M^*$ . The process takes a number of steps, constituting a *work period*.

**Definition 7.1** The parameter  $U = T^*/T$  is defined to be twice the maximum number of steps that any simulation work period can take.  $\lrcorner$

Machine  $M$  will perform the simulation even if the noise in which it operates is  $(\beta(B, T), \gamma(B^*, T^*))$ -sparse. By the above definitions, sparsity means that noise

comes in *bursts* that are confined to rectangles of size  $\beta(B, T)$  (affecting at most  $\beta$  consecutive tape cells), and are separated from each other in time in such a way that there is at most one burst in any  $\gamma$  neighboring work periods. A design goal for the program the following:

**Goal 7.2** (Local correction) *Correct a burst within space and time comparable to its size, and much smaller than the size of a simulation work period.*

There are some difficulties faced by our desire for a structured presentation: In order to analyze the error-correcting performance even of one part of the program, we may need to see the whole, since the noise can change the environment of the head into some state corresponding to an arbitrary other part.

We mentioned *modes* in Section 4.2. Ordinary simulation proceeds in the normal mode. To see whether the basic structure supporting this process is broken somewhere, each step will check whether the current cell-pair is *coordinated* (see Definition 8.6). If not then the state of the current pair will be changed into the *healing* mode. We will also say that *alarm* will be called. On the other hand, the state enters into *rebuilding* mode on some indications that healing fails. (Rebuilding may be triggered inside the Heal rule.)

## 7.1 Head movement

The global structure of a work period is this:

*Computation phase* The new state of the simulated cell pair, and the simulated direction (called the drift) is computed. Then the “meaningfulness” of the result is checked. During this phase the head sweeps, roughly, back-and-forth between the ends of the base colony-pair.

*Transfer phase* The head moves into the neighbor colony in the simulated head direction called drift (building or rebuilding a bridge if needed). Redundancy is used to protect this movement from a burst.

The timing is controlled by a field *Sweep*. We can read off the direction of the sweep  $s$  using the formula

$$\text{dir}(s) = (-1)^{s+1}. \quad (7.1)$$

Some issues complicate the head movement.

### 7.1.1 Zigging

**Definition 7.3** (Front) During normal computation, as the head sweeps in a certain direction, it increases *Sweep* field by 1. Let us call the last site in which this increase occurred, the *front*. ┘

Globally in a configuration, due to disorder, there may be more than one front, but locally we can talk about “the” front without fear of confusion.

A burst could turn the head back in the middle of its sweep, as pointed out in Example 2.2. To detect such an event promptly (in accordance with Goal 7.2), a *zigzag* movement will be superimposed on the sweeping, the head will swing forward and backward from the front. On its backward zig, the head can check the consistency of the last few cells. On its forward zig it can check whether it is not about to enter into forbidden territory. Zigging will use the following parameters:

$$f = 5, \quad Z = c\pi^4 \quad (7.2)$$

for an appropriate constant  $c$ . After every  $f$  steps of progress, the head will perform a forward-backward-forward zigzag of size approximately  $ZB$ , checking the area of size  $ZB$  around the current cell-pair (the turn positions will be modified in Section 7.1.2 below). The fields

$$ZigDir \in \{-1, 0, 1\}, \quad ZigDepth \in [-2Z, 2Z)$$

of the current cell control the process. At the front, we have  $ZigDepth = 0$  and  $ZigDir = 0$ . At the start of zigging, we set  $ZigDir \leftarrow -1$ , and starts *descending* into a (forward) zig, while increasing  $ZigDepth$ . When the front reaches the extreme edge of zigging at an appropriate turning point (see below), we set  $ZigDir \leftarrow 1$ , the head turns towards the front and starts *ascending*, while decreasing  $|ZigDepth|$ . Once the front is reached, a similar backward zigging swing is also performed (with  $ZigDepth$  decreasing below 0).

### 7.1.2 Feathering

Example 2.3 above pointed out the need for the following feature of the simulation program.

**Definition 7.4** (Feathering) A Turing machine execution is said to have the *c-feathering* property if the following holds. If the head turned back at a position  $x$  at some time, then next time it comes within distance  $c$  of  $x$ , it can turn back only at least  $c$  steps beyond  $x$ .  $\lrcorner$

(The name “feathering” refers to the picture of the path of the head in a space-time diagram.) The following example suggests that any computation can be re-organized to accomodate feathering, without too much extra cost.

**Example 7.5** (1-feathering) Suppose that, arriving from the left at position 1, the head decides to turn left again. In repeated instances, it can then turn back at the

following sequence of positions:

1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5, 1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, ...

⌋

If in the original execution the head turned back  $t$  consecutive times to the left from position  $p$ , then now it will turn back from somewhere in a zone of size  $O(\log t)$  to the right of  $p$  in each of these times. (Computing the exact turning point is not necessary.)

We will handle feathering at two kinds of turns: *small* turns and *big* turns. Let

$$F = \pi^4. \quad (7.3)$$

This allows us to define the earlier used parameter

$$PadLen = F \log Q. \quad (7.4)$$

The process will be handled by a field

$$Turned \in \{0, 1, 2\}.$$

The value 1 shows the memory of a recent turn. The value 2 shows the memory of a recent big turn within distance  $F$ . When a cell is passed in which  $Turned > 0$  was found then this value is set to 0.

*Small turns* are done during zigging and healing. Move until you found a cell with  $Turned \neq 1$ , then turn. Set  $Turned \leftarrow 1$  in this cell.

*Big turns* are done during the ends of sweeps in simulation and rebuilding. Move until you found  $F$  consecutive cells with  $Turned < 2$  the last one of which has  $Turned = 0$ , then make a turn. On the cell after the turn, set  $Turned \leftarrow 1$ , and also on the first interval of  $F$  cells after the turn, set  $Turned \leftarrow 2$ .

The control of the above procedures can be accomplished by a counter, say called *FeatherDepth*.

In both cases, there is no limit on the length of the attempt; however, a small turn attempt can take longer than  $O(1)$  steps and a big turn attempt can take longer than  $O(F \log Q)$  steps only in the presence of large disorder (those cases will be analyzed in the context of large disorder). We give here only an informal argument justifying this statement; a formal proof must wait until a complete definition of the simulation and of the notion of large disorder.

For the moment, let us call cells *red* with by  $Turned = 1$  and *orange* with  $Turned = 2$ . Red cells can delay small turns; they can only arise in the forward turn of a zig in the previous sweep. Zigs are by the definition of zigging spaced



by  $f$  cells apart. Big turns are delayed by orange cells; they are also the ones to create orange cells. But they will be organized as in Example 7.5, therefore a big turn attempt will be delayed by at most  $F$  times the logarithm of the total number of big turns at one end of a colony or rebuild area.

The simulated Turing machine will also have the feathering property, therefore the simulation will not turn back from the same colony without passing it in the meantime.

We introduce a convenient notation.

**Notation 7.6** Suppose that within a procedure a turn attempt is made after a move of  $n$  cells in some direction. Then the head may not be able to turn right away, the turn attempt moves it a few more steps in the same direction. In this case we will say that the head moves  $n^+$  cells and turns. Thus  $n^+$  is greater than  $n$ , but by not much: only an amount  $O(1)$  for small turns, and  $O(\frac{1}{2} \log Q)$  for big turns.  $\dashv$

## 7.2 Computation phase

The first phase of the simulation, called the *Compute* rule, computes new values for current cell-pair of the simulated machine  $M^*$  represented by the current (base) colony-pair, and the direction of the move of the head of  $M^*$ . The simulated cell state of  $M^*$  will be represented on the track *Info* of the representing colony. The move direction of  $M^*$  will be represented in the *Drift* field of *each* cell of the base colony-pair (so the whole track must be filled with  $-1$ 's or  $1$ 's). During the execution of this rule, the head sweeps the base colony-pair.

Recall Definition 5.4. The rule *Compute* will rely on a certain fixed  $(\beta, 3)$  burst-error-correcting code, moreover it expects that each of the words found on the *Info* track is 2-compliant. The rule *ComplianceCheck* checks whether a word is 2-compliant.

The rule *Compute* essentially repeats 3 times the following *stages*: decoding, applying the transition, encoding. Then it calls *ComplianceCheck*; if the latter fails it will mark the colony for rebuilding. It uses some additional tracks. A useful one is called

$$Adj'.$$

At the time of the computation, this field of each cell of the left base colony should contain a single symbol in  $\{0, 1\}$ , according to whether the previous left base colony was adjacent or not. A track called *Work* can be used for auxiliary computation. In more detail:

1. For every  $j = 1, \dots, 3$ , do

- a. Calling by  $\mathbf{a}$  the pair of strings found on the *Info* track of the interior of the base colonies, decode it into the pair of strings  $\tilde{\mathbf{a}} = \nu^*(\mathbf{a})$  (this should be the current state of the simulated cell-pair), and store it on some auxiliary track in the base colony-pair. Do this by simulating the universal machine on the *Prog* track:  $\tilde{\mathbf{a}} = \text{Univ}(p_{\text{decode}}, \mathbf{a})$ . However, then replace  $\tilde{a}_0.\text{Adj}$  with the value found on the  $\text{Adj}'$  track.
  - b. Compute the value  $(\mathbf{a}', d) = \tau^*(\tilde{\mathbf{a}})$ . Since the program of the transition function  $\tau^*$  is not written explicitly anywhere, this “self-simulation” step needs some elaboration, see Section 7.3.
  - c. Write the encoded new cell states  $\nu_*(\mathbf{a}')$  onto the  $\text{Hold}[j].\text{Info}$  track of the interior of the base colony-pair. Write  $d$  into the  $\text{Hold}[j].\text{Drift}$  field of *each cell* of the left base colony.  
 Special action needs to be taken in case a new state  $a'_0$  or  $a'_1$  is a vacant one, that is for example  $a'_0.\text{Kind}^* = \text{Vac}^*$ . In this case, write 1 onto the  $\text{Hold}[j].\text{Doomed}$  track (else 0).
2. Sweeping through the base colony pair, at each cell compute the majority of  $\text{Hold}[j].\text{Info}$ ,  $j = 1, \dots, 3$ , and write it into the field *Info*. Proceed similarly, and simultaneously, with *Drift*.
  3. For  $j = 1, \dots, 3$ , call *ComplianceCheck* on the *Info* track, and write the resulting bit into the  $\text{Compliant}_j$  track.  
 Then pass through the colony and turn each cell in which the majority of  $\text{Compliant}_j$ ,  $j = 1, \dots, 3$  is false, into a stem cell (thus destroying the colony if the result was false everywhere).  
 It can be arranged—and we assume so—that the total number of sweeps of this phase, and thus the starting sweep number of the next phase, depends only on  $Q$ .

## 7.3 Forced self-simulation

Step 1b of Section 7.2 needs elaboration.

### 7.3.1 New primitives

We will make use of a special track

*Work*

of the cells and the special field

*Index*

that can store a certain address of a colony.

Recall from Section 4.2 that the program of our machine is a list of nested “if condition **then** instruction **else** instruction” statements. As such, it can be represented as a binary string

$$R.$$

If one writes out all details of the construction of the present paper, this string  $R$  becomes completely explicit, an absolute constant. But in the reasoning below, we treat it as a parameter.

Let us provide a couple of *extra primitives* to the rules. First, they have access to the parameter  $k$  of machine  $M = M_k$ , to define the transition function

$$\tau_{R,k}(\mathbf{a}).$$

The other, more important, new primitive is a special instruction

WriteProgramBit

in the rules. When called, this instruction makes the assignment  $Work \leftarrow R(Index)$ . This is the key to self-simulation: *the program has access to its own bits*. If  $Index = i$  then it writes  $R(i)$  onto the current position of the *Work* track.

### 7.3.2 Simulating the rules

By convention, in our fixed flexible universal machine Univ, program  $p$  and input  $x$  produce an output  $Univ(p, x)$ . Since the structure of all rules is very simple, they can be read and interpreted by Univ in reasonable time:

**Theorem 2** *There is a constant string called *Interpr* with the property that for all positive integers  $k$ , string  $R$  that is a sequence of rules, and a pair of bit strings  $\mathbf{a} = (a_0, a_1)$  with  $a_j \in \Sigma_k$ ,*

$$Univ(Interpr, R, 0^k, \mathbf{a}) = \tau_{R,k}(\mathbf{a}).$$

*The computation on Univ takes time  $O(|R| \cdot |\mathbf{a}|)$ .*

The proof parses and implements the rules in the string  $R$ ; each of these rules checks and writes a constant number of fields.

Implementing the WriteProgramBit instruction is straightforward: Machine Univ determines the number  $i$  represented by the simulated *Index* field, looks up  $R(i)$  in  $R$ , and writes it into the simulated *Work* field.

Note that there is no circularity in these definitions:

- The instruction `WriteProgramBit` is written *literally* in  $R$  in the appropriate place, as “`WriteProgramBit`”. The string  $R$  is *not part* of the rules (that is of itself).
- On the other hand, the computation in  $\text{Univ}(\text{Interpr}, R, 0^k, \mathbf{a})$  has *explicit* access to the string  $R$  as one of the inputs.

Let us show the computation step invoking the “self-simulation” in detail. In the earlier outline, step [1b](#) of Section [7.2](#), said to compute  $\tau^*(\tilde{\mathbf{a}})$  (for the present discussion, we will just consider computing  $\tau^*(\mathbf{a}) = \tau_{k+1}(\mathbf{a})$ ), where  $\tau = \tau_k$ , and it is assumed that  $\mathbf{a}$  is available on an appropriate auxiliary track. We give more detail now of how to implement this step:

1. Onto the *Work* track, write the string  $R$ . To do this, for *Index* running from 1 to  $|R|$ , execute the instruction `WriteProgramBit` and move right. Now, on the *Work* track, replace it with  $\langle \text{Interpr}, 0^{k+1}, R, \mathbf{a} \rangle$ . Here, string *Interpr* is a constant, so it is just hardwired. String  $R$  already has been made available. String  $0^{k+1}$  can be written since the parameter  $k$  is available. String  $\mathbf{a}$  is available on the track where it is stored.
2. Simulate the universal automaton  $\text{Univ}$  on track *Work*: it computes  $\tau_{R,k+1}(\mathbf{a}) = \text{Univ}(\text{Interpr}, R, 0^{k+1}, \mathbf{a})$  as needed.

This achieves the forced self-simulation. Note what we achieved:

- On level 1, the transition function  $\tau_{R,1}(\mathbf{a})$  is defined completely when the rule string  $R$  is given. It has the forced simulation property by definition, and string  $R$  is “*hard-wired*” into it in the following way. If  $(\mathbf{a}', d) = \tau_{R,1}(\mathbf{a})$ , then

$$a'_0.\text{Work} = R(a_0.\text{Index})$$

whenever  $a_0.\text{Index}$  represents a number between 1 and  $|R|$ , and the values  $a_0.\text{Sweep}$ ,  $a_0.\text{Addr}$  satisfy the conditions under which the instruction `WriteProgramBit` is called in the rules (written in  $R$ ).

- The forced simulation property of the *simulated* transition function  $\tau_{R,k+1}(\cdot)$  is achieved by the above defined computation step—which *relies on* the forced simulation property of  $\tau_{R,k}(\cdot)$ .

**Remark 7.7** This construction resembles the proof of Kleene’s fixed-point theorem. ┘

## 7.4 Transfer phase

In the transfer phase the `Transfer` rule takes over: control will be transferred to the neighbor colony in the direction of the simulated head movement: this is

called the direction of the transfer, or the *drift*. As we will see the transfer phase takes only a constant number of sweeps: three pairs of sweeps should be sufficient (see the details below). During this phase, the range of the head includes the base colony pair and a neighbor colony in the direction of the drift called *target colony*, including possible bridges between them. The bridges present some extra complication; let us address it.

The sweep number in which we start transferring in direction  $\delta$  is called  $\text{TransferSw}(\delta)$ , the *transfer sweep*. We have  $\text{TransferSw}(-1) = \text{TransferSw}(1) + 1$ .

**Definition 7.8** (Adjacency of cells) Cells  $a$  and  $b$  are *adjacent* if  $|a - b| = B$ . Otherwise, if  $B < |a - b| < 2B$ , then  $a$  and  $b$  are two *non-adjacent neighbor cells*. For the sake of the present discussion, a *colony* is a sequence of  $Q$  adjacent cells whose *Addr* value runs from 0 to  $Q - 1$ . It may be *extended* by a bridge of up to  $Q - 1$  adjacent cells.

If the bodies of two cells are not adjacent, but are at a distance  $< B$  then the space between them is called a *small gap*. We also call a small gap such a space between the bodies of two colonies. On the other hand, if the distance of the bodies of two colonies is  $> B$  but  $< QB$  then the space between them is called a *large gap*.  $\lrcorner$

Before the transfer phase, the base colony pair consists of two colonies, having cells of kind  $\text{Member}_0$  and  $\text{Member}_1$ . There may be a bridge between them, of  $< Q$  cells of kind *Bridge* adjacent to each other, and extending one of the base colonies (see Definition 4.6 and the remark following it).

The transfer phase moves the base colony pair left, if  $\text{Drift} = -1$ , and right, if  $\text{Drift} = 1$  in the left base colony.

Consider  $\text{Drift} = -1$  first. The kind of cells of the left base colony will turn from  $\text{Member}_0$  to  $\text{Member}_1$ . Then a bridge is built to the left (extending the  $-1$  value on the *Drift* track). This bridge can override an opposite old bridge (“old” meaning that its *Sweep* is maximal) or move into an empty area, kill other bridge cells or stem cells if they are not adjacent, while it extends. There are two ways that this bridge extension can finish.

- It stops at the boundary of another colony on the left (which consists of cells of type  $\text{Member}_0$ ). (This can happen in the very first step, in which case no bridge is built.) Then this colony becomes the new left base colony, and the head moves to its left end (extending again the  $-1$  value on the *Drift* track).
- The bridge reaches the size of  $Q$  cells. In this case, the bridge will be converted into the new left base colony, the kind of its cells becoming  $\text{Member}_0$ .

Recall that the *Adj* field determines whether the current cell is adjacent to the cell where the head came from. After the transfer stage, we write the *Adj'* track of the

new left base colony: it becomes 0 if either there is a nonempty bridge, or there is a gap (found with the help of the *Adj* field) between the old and new left base colony. This is done again three times, storing candidate values into  $Hold[j].Adj'$ , repeating and doing a majority vote.

In the last sweep, in the old base colony pair, if the majority of  $Hold[j].Doomed$ ,  $j = 1, \dots, 3$ , is 1 then turn the scanned cell into a stem cell: in other words, carry out the destruction.

The case  $Drift = 1$  is analogous, with the following logical changes. First the bridge between the base colonies is possibly rebuilt, to extend the left base colony. Then the kind of cells of the right base colony will turn from  $Member_1$  to  $Member_0$ , and it becomes the new left base colony. Then a bridge is build to its right (extending the 1 value on the *Drift* track). It stops at the boundary of another colony on the right (which consists of cells of type  $Member_1$ ), or becomes itself a colony with elements having kind  $Member_1$ .

## 8 Health

The main part of the simulation uses an error-correcting code to protect information stored in *Info* field. However, faults can ruin the simulation structure and disrupt the simulation itself. The error-correcting capabilities of the code will preserve the content of the *Info* track as long as the coding-decoding process implemented in the simulation is carried out. The correctness of the process itself relies only on the structural integrity of a configuration; this in its turn is maintained with the help of a small number of fields. Below we outline the necessary relations among them allowing the identification and correction of local damage.

A configuration with local structural integrity will be called *healthy*.

**Remark 8.1** In all discussions of the health of a configuration  $\xi = (A, h, \hat{h})$ , we can ignore the current cell position  $\hat{h}_0 = \xi.cur-cell$ , since at all switching times it agrees with  $h = \xi.pos$ .  $\lrcorner$

All configurations are made up of homogenous segments.

**Definition 8.2** (Homogenous segment) A segment of non-stem adjacent neighbor cells is *homogenous* if its addresses are increasing from left to right, and its members have the same value of *Kind*, *Sweep*, *Drift*, *Adj'*. The *left end* of a segment is the left edge of its first cell, and its *right end* is the right edge of its last cell.  $\lrcorner$

A key property of our definition of health is that it is completely determined by the types of segments allowed and the types of boundaries allowed between them. This allows the local detection of failures of health. The big picture is

that cells of a healthy configuration are grouped into gapless colonies, with certain transitional segments inside and between them.

- There is a base colony-pair, with possibly a bridge between them going in the direction of its own drift. If the sweep inside this colony-pair shows that the Compute rule is being performed then the sweep boundaries allowed inside the colony-pair are the front, and the edges according to past turns at the end of the interior of the base colony-pair. Outside each edge is a segment with a lower sweep number.
- During the first sweep of transfer, the base colony (of the pair) in the direction of the drift is possibly extended by a bridge towards the target colony. This bridge may be in the process of consuming an earlier bridge extended in the opposite direction. In the later part of the first sweep of transfer, the bridge already reaches the target colony.

If the bridge extends to a full colony then this is converted to the corresponding kinds of member cells in the second sweep of the transfer.

In the later sweeps of the transfer, segments ahead and behind the front show the changes done by the transfer phase, including the change of  $Adj'$  in the last sweep.

- Non-base colonies are called *outer colonies*. If an outer colony is not adjacent to its neighbor colony closer to the base, then it is extended by a bridge that covers this gap.

## Peter remark 2. Pictures! ┘

Let us define outer cells and boundary types in more detail.

**Definition 8.3** (Outer cells, desert) Recall the definition of the sweep value  $Last(\delta)$  from (4.1). For  $\delta \in \{-1, 1\}$ , if a cell is stem or has  $Drift = \delta$ ,  $Sweep = Last(\delta)$  then it will be called a *right outer cell* if  $\delta = -1$ , a *left outer cell* if  $\delta = 1$ . ┘

According to this definition, a stem cell is both a left and a right outer cell. Recall Definition 7.3 of the front.

**Definition 8.4** (Boundaries) A boundary of a homogenous segment is called *rigid* if its address is the end address of a colony in the same direction. A *boundary pair* is a right boundary followed by a left boundary at distance  $< B$ . It is a *hole* if the distance is positive. It is *rigid* if at least one of its elements is. ┘

The health of a configuration  $\xi$  of a generalized Turing machine  $M$  is defined over a certain interval  $A$ . It depends on  $\xi_{\text{tape}}(A)$ , further on whether  $\xi_{\text{pos}}$  is in  $A$  and if it is, where. But we will mention the interval  $A$  explicitly only where it is

necessary. In particular, some of the structures described above may fall partly or fully outside  $A$ .

Defining health formally can be done mechanically on the basis of the above informal description, but the detailed description would be tedious.

A tape configuration is called *healthy* on an interval  $A$  when there is a head position (possibly outside  $A$ ) and a state that turns it into a healthy configuration.

**Definition 8.5** The tuple of fields

$$Core = (Addr, Sweep, Drift, Kind)$$

is called the *core*. ┘

Health only depends on the state, including the zigging field  $Z$ , of the current cell, further the *Core* track and the lack of heal or rebuild marks on the tape. Note that in a healthy configuration every cell's *Core* field determines the direction in which the front is found, from the point of view of the cell.

A violation of the health requirements can sometimes be noted immediately:

**Definition 8.6** (Coordination) The current cell pair is *coordinated* if it is possible for them to be together in a healthy configuration. ┘

In normal mode, if lack of coordination is discovered then the healing procedure is called. The following lemmas show how local consistency checking will help achieve longer-range consistency.

**Lemma 8.7** *In a healthy configuration, each Core value along with  $Z$  determines uniquely the Core value of the other cell of the current cell pair, with the following exceptions.*

- *During the first transferring sweep, while creating a bridge, the front can be a stem cell or the first cell of an outer colony.*
- *Every jump backward from the target colony can end up on the last cell of a bridge (whose address is not recorded in the cell from which the head is jumping) or the last cell of the base colony.*

*Proof.* To compute the values in question, calculate  $Z$  steps backwards from the front, referring to the properties listed above.

**Peter remark 3.** Elaborate! ┘

□

**Lemma 8.8** (Health extension) *Let  $\xi$  be a tape configuration that is healthy on intervals  $A_1, A_2$  where  $A_1 \cap A_2$  contains a whole cell body of  $\xi$ . Then  $\xi$  is also healthy on  $A_1 \cup A_2$ .*



*Proof.* The statement follows easily from the definitions.

**Peter remark 4.** Elaborate! ┘

□

In a healthy configuration, the possibilities of finding non-adjacent neighbor cells are limited.

**Lemma 8.9** *An interval of size  $< Q$  over which the configuration  $\xi$  is healthy contains at most two maximal sequences of adjacent non-stem neighbor cells.*

*Proof.* Indeed, by definition a healthy configuration consists of full extended colonies, with possibly stem cells between them. An interval of size  $< Q$  contains sequences of adjacent cells from at most two such extended colonies. □

**Lemma 8.10** *In a healthy configuration, a cell's content shows whether it is an outer cell, colony cell, bridge cell or target colony cell. It also shows whether the cell is to the left or to the right of the front.*

*The Core track of a homogenous segment can be satisfactorily reconstructed from any of its cells. The reconstruction is unique, with one exception: in case the segment is not internal, the Sweep values are not uniquely defined, only the parity and the direction of its change between the ends.*

*Proof.* The information mentioned in the first sentence is explicit in some fields. About the front: if the cell is outer then *Drift* shows its direction from the front. In case it is not outer, then the parity of *Sweep* shows it: odd values are on the left, even ones on the right.

The uniqueness of reconstruction is a direct consequence of the definitions. □

## 9 Healing and rebuilding

### 9.1 Annotation, admissibility, stitching

In this section we show how to correct configurations of machine  $M$  that are “almost” healthy.

**Definition 9.1** (Annotation) *An annotated configuration is a tuple*

$$(\xi, \chi, \mathcal{I}),$$

with the following meaning.

$\xi$  is a configuration.

$\mathcal{I}$  is a set of disjoint intervals called *islands*. Their complement is clean.

$\chi$  is a healthy configuration differing from  $\xi$  only in the islands.

The *base colony* of  $(\xi, \chi, \mathcal{I})$  is that of  $\chi$ . The head is *free* when it is not in any island, and the observed cell is in normal mode, coordinated with its pair.  $\lrcorner$

**Definition 9.2** (Admissibility) An annotation is *admissible* on an interval  $K$  if the following holds on any subinterval  $J$  of  $K$  of size  $QB$ :

- a1) The disorder of  $J$  is covered by 3 intervals, each of size  $\leq \beta'B$ .
- a2) There are at most 3 islands in  $J$ , each of size at most  $c_{\text{island}}\beta B$ .

A configuration is *admissible* on  $K$  if it can be annotated in a way admissible on  $K$ .  $\lrcorner$

We will show that a configuration admissible over an interval of a certain size can be locally corrected; moreover, in case the configuration is clean then this correction can be carried out by the machine  $M$  itself. For the time being, we will just talk about an admissible configuration, without specifying the interval  $K$ .

We will deal with the cleaning process later, but it will imply that, in the absence of new noise, islands will not grow much, and the ones near the head will be eliminated.

**Definition 9.3** (Substantial segments) Let  $\xi(A)$  be a tape configuration over an interval  $A$ . A homogenous segment of size at least  $7c_{\text{island}}\beta B$  will be called *substantial*. The area between two neighboring maximal substantial segments or between an end of  $A$  and the closest substantial segment in  $A$  will be called *ambiguous*. It is *terminal* if it contains an end of  $A$ . Let

$$\Delta = 39c_{\text{island}}\beta,$$

$$E \geq 6\Delta.$$

$\lrcorner$

**Lemma 9.4** Consider an admissible configuration. In a substantial segment, each half contains at least one cell outside the islands.

If an interval of size  $\leq QB$  of a tape configuration  $\xi$  differs from a healthy tape configuration  $\chi$  in at most three islands, then the size of each ambiguous area is at most  $\Delta B$ .

*Proof.* The first statement is immediate from the definition of substantial segments. There are at most 3 boundary pairs in  $\chi$  at a total size of  $3B$ , and 3 islands of size  $\leq c_{\text{island}}\beta B$ . There are at most 5 non-substantial segments of sizes  $< 7c_{\text{island}}\beta B$  between these: this adds up to

$$< (3 + 3 \cdot c_{\text{island}}\beta + 5 \cdot 7 \cdot c_{\text{island}}\beta)B < 39 \cdot c_{\text{island}}\beta B = \Delta B.$$

□

The following lemma forms the basis of the cleaning algorithm.

**Lemma 9.5** (Stitching) *In an admissible configuration, inside a clean interval, let  $U, W$  be two substantial segments separated by an ambiguous area  $V$ . It is possible to change the tape on  $U, V, W$  using only information in  $U, W$  in such a way that the tape configuration over  $U \cup V \cup W$  becomes healthy. Moreover, it is possible for a cellular automaton to do so gradually, keeping admissibility, and changing the tape in  $U$  or  $W$  or enlarging  $U$  or  $W$  gradually at the expense of  $V$ .*

We do not worry about the feathering property now, since the machine in question can choose where to make its turns.

*Proof.* We distinguish several cases, based on the kind of segments involved. At any step, if we find that  $U \cup V \cup W$  is healthy then we stop.

1. Assume that  $U, W$  both belong to the same extended colony: either an extended base colony, or an outer extended colony, or the target.

If both belong to the interior of the area indicated, then the merging is simple. In this case, by Lemma 8.10, the content of the *Core* track of  $V$  in the healthy configuration is completely determined by that of  $U, W$ . So the intervals  $U$  and  $W$  can be gradually extended towards each other in any order, overtaking  $V$ .

Suppose that they do not belong to the interior and are, for example towards the right from it. In this case, the *Sweep* value may be decreasing in both  $U$  and  $W$  to the right. But since  $U, W$  themselves may overlap with islands, it may happen that the *Sweep* value on the right end of  $U$  is smaller than the *Sweep* value on the left end of  $W$ . In this case, before extending  $U, W$  towards each other, the *Sweep* values in both may need to be changed.

By Lemma 9.4, the left and right halves of  $U$  contains a cell  $u_1, u_3$  each, not belonging to any island, and similarly with  $w_1, w_3$  in  $W$ . Let  $u_2$  be leftmost cell of the right half of  $U$  and  $w_2$  the rightmost cell of the left half of  $W$ . Then

$$\text{Sweep}(u_1) \geq \text{Sweep}(u_2) \geq \text{Sweep}(u_3) \geq \text{Sweep}(w_1) \geq \text{Sweep}(w_2) \geq \text{Sweep}(w_3).$$

So the transformation will change *Sweep* to  $\text{Sweep}(u_2)$  everywhere in the right half of  $U$ , and to  $\text{Sweep}(w_2)$  everywhere in the left half of  $W$ . After this, it will extend  $U$  towards  $W$ , overtaking  $V$  and keeping *Sweep* constant.

In both of the above cases, if  $U$  and  $W$  belonged to different sides of the front, then the new front will be the new boundary between  $U$  and  $W$ .

2. Suppose now that  $U, W$  do not belong to the same extended colony, but they are on the same side of the front: without loss of generality, to the left of it.

In this case, only  $U$  can be outer: suppose that it is. Now  $W$  is either in a target or in the base colony. The base colony cannot have an extension to the left, because given that the front is to the right, the same sweep that created the extension would have created already a target, separating  $U$  from  $W$  too much. So in both cases,  $W$  is close to its colony's the left end, which is in  $V$ . It must be extended to this left end. Following this,  $U$  must be extended until it reaches  $W$ . The *Sweep* values can be extended without change, there is no need to coordinate them between  $U$  and  $W$ .

Suppose that  $U$  is a target, and then  $W$  belongs to the extended base colony. Since both  $U$  and  $W$  are in the interior, the *Sweep* values must be equal, so they can be extended without change. Then  $U$  must be extended until its endcell, and then  $W$  must be extended to meet  $U$ . Since both  $U$  and  $W$  are in the interior, the *Sweep* values must be equal, so they can be extended without change.

3. Suppose that  $U$  and  $W$  belong to different extended colonies, with the front between them.

The *Sweep* values can be extended without change, there is no need to coordinate them between  $U$  and  $W$ .

Suppose that  $U$  contains no bridge cells: in this case extend it to the right until it hits a colony endcell. If you overlap  $W$ , decrease  $W$  accordingly. Due to admissibility, only bridge cells of  $W$  can be overwritten in this way (colonies don't overlap in a healthy configuration). Similarly, if  $W$  contains no bridge cells then extend it to the left, until it hits a colony endcell. Again, only bridge cells of  $U$  can be overwritten in this way.

Only one of  $U$  and  $W$  can be in an outer extended colony, suppose that  $U$  is. If  $W$  is a target then we already extended it to its limit. Now extend the bridge of  $U$  to meet it. If  $W$  is in an extended base colony then extend its bridge until either meets  $U$  or reaches full length. In the latter case, extend the bridge of  $U$  to meet  $W$ .

If  $U$  is in a target then it is already at its right end: we extend the bridge of  $W$  to meet it.

□

## 9.2 The healing procedure

Structure repair will be split into two procedures. The first one, called *healing*, performs only local repairs of the structure: for a given (locally) admissible configuration, it will attempt to compute a satisfying (locally) healthy configuration. If it fails—having encountered a configuration that is not admissible, or a new

burst—then the *rebuilding* procedure is called, which is designed to repair a larger interval. On a higher level of simulation, this corresponds to the implementation of the “cleaning” trajectory properties. The healing procedure runs in  $O(\beta)$  steps, whereas rebuilding needs  $O(Q^2)$  steps.

**Peter remark 5.** maybe even  $Q^3$ ? ┘

The description of the procedures looks as if we assumed that there is no noise or disorder. The rules described here, however (as will be proved later), will clean an area locally under the appropriate conditions, and will also work under appropriately moderate noise.

The healing procedure does not even protect itself against any possible noise during it. The only protection is that any one call of the healing procedure will change only a small part of the tape, essentially one cell: so a noise burst will have limited impact even if it happens during healing.

One possible outcome of the healing procedure is *failure*. In this case the plan is to mark a “germ” of  $4\beta$  cells and call rebuilding. The healing procedure carries out only one step of this plan, then it calls itself again.

Recall the parameters  $\Delta, E$  introduced in Definition 9.3. Suppose that *Heal* is called at some position  $z$ . Then it sets

$$Mode \leftarrow \text{Healing}, \text{Heal.Sweep} \leftarrow 1, \text{Heal.Addr} \leftarrow 0.$$

The healing procedure starts by surveying an interval  $R$  of at least  $2E$  cells around its starting point  $z$ . (It will go a little farther than  $E$  cells, in search of an allowed turning point, due to feathering. If such a point is not found within  $3\beta$  steps, healing fails.) Whenever the head steps on a stem cell or creates a new cell, *Drift* is set to point to  $z$  (to make sure that the head does not get lost in a homogenous segment of stem cells).

Suppose that in the survey a pair of substantial segments separated by an ambiguous area is found, that is at least  $\Delta$  removed from the complement of  $R$ . Choose the ambiguous area closest to the center (of healing). Then the first needed “stitching” operation (a single step) as defined in Lemma 9.5 is determined, and is performed. If the ambiguous area still remains, go to its cell closest to  $z$  and restart healing. In case that we started from a clean admissible configuration, this operation creates a new admissible configuration that is one step closer to being healthy.

If the required stitching operation is not found then, the surveyed area is found inadmissible, and healing fails. Otherwise, if the head is at the front, healing is declared completed. Otherwise, the healing center is moved one step closer to the front, and healing is restarted.

The healing operation defined this way has the following property.

**Lemma 9.6** *Assume that the head moves in a noise-free and clean space-time rectangle  $[a, b) \times [u, v)$ , with  $b - a > 3EB$ ,  $v - u > 2ET$ , touching every cell of  $[a, b)$  at least once, and never in rebuilding mode. Then at time  $v$ , the area  $[a + 1.5EB, b - 1.5EB)$  is healthy.*

*Proof.* In healing, the head will not move away from an ambiguous area before eliminating it, creating a healthy interval  $I$  containing the two substantial intervals that originally bordered it. If the head leaves this interval in normal mode and the zigging does not start new healing then the healthy interval covered by zigging can be added to  $I$  to form an even larger healthy interval, and this continues until new healing starts. One of the substantial intervals of the new healing will be inside  $I$ , and when it is completed,  $I$  will be extended further. The head may later return into the healthy interval  $I$ , but it will then just continue the simulation without affecting health while staying inside.  $\square$

### 9.3 Rebuilding

If healing fails, it calls the rebuilding procedure. This indicates that the colony structure is ruined in an interval of size larger than what can be handled by local healing. Just as with healing, we will be speaking here only about a situation with no mention of disorder—but the final analysis will take disorder into account.

Since rebuilding makes changes on an interval of the length of several colonies, it is important not only that it is invoked when it is needed, but that it is not invoked otherwise! A failed healing ends on a “germ” of cells marked for rebuilding, with the state in rebuild mode. Rebuilding starts by extending the germ to the left, in a zigging way, expecting rebuild-marked cells on the backward zig. Since the zig is larger than a burst, if rebuilding started from just a burst then this will trigger healing, thus leaving the rebuilding mode.

The notion of front, of Definition 7.3 will be extended also to the rebuilding mode.

Here is an outline, for rebuilding that started from a cell  $z$  in the middle of the germ. The goal is that

- the process does not destroy any healthy colony more than  $3\beta B$  to the left of  $z$  or to the right of  $z$ .
- we end up with a new decodable area (see Definition 10.4) extending at least one colony to the left and one colony to the right of  $z$ .

Recall the stitching operation from Section 9.1.

*Mark* Starting from the germ, extend a rebuilding area over  $3Q$  cells to the left and  $3Q$  cells to the right from  $z$ . Mark the area using the track  $Rebuild.Addr_j$  for  $j = \pm 1$ , where addresses are counted from both ends of the rebuilding area, and

the track *Rebuild.Sweep*. False rebuilding started by a burst will be recognized since zigging must see an at least germ-sized marked area. In what follows, every step that changes the configuration must be accompanied by zigging, to check that the rebuilding is indeed going on.

*Survey and Create* More details of this stage will be given below. It looks for existing colonies (possibly needing minor repair) in the rebuilding area, and possibly creates some. As a result, we will have one colony called  $C_{\text{left}}$  on the left of  $z$ , one called  $C_{\text{right}}$  on the right of  $z$ , and possibly some colonies between them. Make all newly created colonies represent stem cells. Declare  $C_{\text{left}}$  the base colony, direct all the others with drifts and bridges towards it. (The creation of a bridge may result also in the creation of a new colony if the bridge becomes  $Q$  cells long.) The interval covering  $C_{\text{left}}$  and  $C_{\text{right}}$  will be called the *output interval* of rebuilding.

*Mop* Remove the rebuild marks, shrinking the rebuilding area onto the left end of  $C_{\text{left}}$ .

#### Details of the Survey and Create stage

- s1) Search in the marked area on left of  $z$  for a colony  $C$ , or a set of cells that looks stitchable by up to  $3$  healings into a colony. The first substantial segment should be at least  $3\beta$  cells to the left of  $z$ , to make sure that  $C$  is *manifestly* to the left of  $z$ . If the search and the stitching attempt are successful, mark  $C$  as  $C_{\text{left}}$ .  
Repeat this search for a colony manifestly to the right of  $z$ : if found, call it  $C_{\text{right}}$ .
- s2) Suppose that only  $C_{\text{left}}$  is found, then create  $C_{\text{right}}$ . The other case is symmetrical.
- s3) Suppose that neither  $C_{\text{left}}$  nor  $C_{\text{right}}$  have been found. Then search the whole area, starting from the left, for a colony. If no such colony is found then create  $C_{\text{left}}$  and  $C_{\text{right}}$ .
- s4) Suppose that both  $C_{\text{left}}$  and  $C_{\text{right}}$  have been determined (found or created). Then search between them, from the left, for (stitchable) colonies, one-by-one.
- s5) Suppose that only a colony has been found that is not manifestly to the left or right of  $z$ ; then either the left end is manifestly to the left or the right end is manifestly to the right of  $z$ .  
Suppose that the left end it is manifestly to the left of  $z$ , then call the colony  $C_1$ . Then create  $C_{\text{left}}$  on the left of  $C_1$ . Now search for another one on its

right (it is not manifestly on the right). If not found, create  $C_{\text{right}}$ , manifestly on the right of  $z$ . If found call it  $C_2$ , and create  $C_{\text{right}}$  on its right. The other case is symmetrical.

The steps above requiring the creation of colonies and bridges are destructive (the stitching ones are not). Therefore before a creation step, the whole survey preceding it is performed twice, marking the result in all rebuilding cells. The required actions (erasing cells in order to build new ones in their place) are then only performed if the two survey results are identical—otherwise alarm is called.

To defend against the effects of a burst during the rebuild procedure, any decision on creation is made twice. The two surveys on which the decision is based note their results everywhere on two different tracks. These results must agree, otherwise rebuilding fails and new healing (followed probably by new rebuilding) is started. (Marked cells from some interrupted rebuilding may remain even after the mop-up operation. These may trigger new healing-rebuilding later.)

## 10 Scale-up

### 10.1 Annotated history

It is convenient to introduce some additional structures when discussing the effects of moderate noise and their repair.

The following definitions help to extend the notion of annotation to histories.

**Definition 10.1** (Distress and relief, safety) Consider a sequence of annotated configurations over a certain time interval. If the head is free (see Definition 9.1), then the time (and the configuration) will be called *distress-free*. A time that is not distress-free and is preceded by a distress-free time will be called a *distress event*. This can be of two kinds: the head steps onto an island, or a burst occurs (creating an island and leaving the head in it).

Consider a time interval  $K$  starting with a distress event and ending with a distress-free configuration. Let  $J$  be the interval of tape where the head passed during  $K$ , then we will call  $J \times K$  a *relief event*, if the following holds.

- a) Any new islands occurring in  $J \times K$  are due to some new burst.
- b) The island that started the distress event disappears by the end of  $K$ .

┘

In a relief event, it is possible to leave behind some islands other than the one initiating the distress.

**Example 10.2** (Three islands) Suppose that the head has arrived at  $C$  from the left, performs a work period and then passes to the right. In this case, if no new



noise burst occurs then we expect that all islands found in  $C$  will be eliminated by the healing procedure. On the other hand, a new island  $I_1$  can be deposited (in the last pass). We can assume that there is no island on the left of  $I_1$  within distance  $QB$ , since the noise burst causing it would have been too close to the noise burst causing  $I_1$ .

Consider the next time (possibly much later), when the head arrives (from the right). If it later continues to the left, then the situation is similar to the above. Island  $I_1$  will be eliminated, but a new one may be deposited. But what if the head turns back right at the end of the work period? If  $I_1$  is close to the left end of  $C$ , then due to the feathering construction, the head may never reach it to eliminate it; moreover, it may add a new island  $I_2$  on the right of  $I_1$ . We can also assume, similarly to the above, that there is no island on the right of  $I_2$  within distance  $QB$ .

When the head returns a third time (possibly much later), from the right, it will have to leave on the left. The islands  $I_1, I_2$  will be eliminated as they are passed over but a possible new island  $I_3$ , created by a new burst (before, after or during the elimination), may remain. We can also assume, similarly to the above, that there is no island on the right of  $I_3$  within distance  $QB$ .

In the work period where the head deposits island  $I_2$  near the left colony end, it may repeatedly dip into  $I_2$  at the descending end of a zig at a right turn. During this dip it can expand the islands  $I_1, I_2$  and then emerge on the right, with the healing unfinished.  $\lrcorner$

We will consider the annotation of histories over a limited space-time region, but will not point this out repeatedly.

**Definition 10.3** (Annotated history) An *annotated history* of a generalized Turing machine

$$M = (\Sigma, \tau, Adj, B, T, \pi)$$

is a sequence of super-annotated configurations such that the sequence of underlying configurations is a trajectory, and every distress event is followed by a relief event  $J \times K$  with  $|J| = O(\beta B)$  and  $|K| = O(\beta^2 T)$ .  $\lrcorner$

In what follows we will show that for any trajectory  $(\eta, Noise)$  of a generalized Turing machine  $M$  on any space-time rectangle on which the noise is  $(\beta(B, T), \gamma(B^*, T^*))$ -sparse, if at the beginning the configuration was super-healthy then the history can be annotated. In the rest of the section we always rely on the assumption of this sparsity property of the noise.

The main part of the proof is about obtaining relief after a distress event. Unlike in [1], now islands may have their cell structure damaged: may contain disorder. However, since  $\eta$  is a trajectory, as we will see the islands will be cleaned out.

So, relief will happen in two stages: cleaning, and correcting the structure. This division is only possible for an observer: the machine has no “disorder-detector”, we just rely on the cleanness-extending properties of a trajectory introduced in Definition 6.10.

## 10.2 Super-health

**Definition 10.4** (Super-health) Let  $\xi$  be a clean configuration on an interval  $I$ . We say that  $\xi$  is *super-healthy* if both ends of  $I$  coincide with the end of a colony, and in each colony in  $I$ , whenever the head is not in the last sweep, the *Info* contains a valid codeword as defined in Section 5.2. A tape configuration  $\xi$  is *super-healthy* if it can be extended to a super-healthy configuration.  $\dashv$

As indicated in Section 6, when dealing with the behavior of machine  $M$  over some space-time rectangle, we will assume that the noise over this rectangle is  $(\beta(B, T), \gamma(B^*, T^*))$ -sparse. With Definition 7.1 of  $T^*$  this means in simpler terms that at most one noise *burst* affecting an area of size at most  $\beta B$  can occur in any  $\gamma$  consecutive work periods. In the present section, histories will always be assumed to have this property.

Some histories lend themselves to be viewed as a healthy development that is disturbed only in some well-understood ways. Added to such histories the information pointing out these disturbances will be called an annotation. The proof of the error-correcting behavior of machine  $M$  (essentially the proof of the Transition Function property of trajectories of Definition 6.10 for the simulated machine  $M^*$ ) will take the form of showing the possibility of annotation under sparse noise. An annotation, as per Definition 9.1, marks some ways in which the health of a tape configuration has been affected. Now we extend annotation in order to deal with damage not only to health but also to information.

**Definition 10.5** (Super-annotation) A *super-annotated configuration* is a tuple

$$(\xi, \chi, \mathcal{I}, \mathcal{S}),$$

with the following meaning.

$(\xi, \chi, \mathcal{I})$  is an annotated configuration.

$\mathcal{S}$  is a set of disjoint intervals called *stains*. All islands are contained in stains.

We can change  $\chi$  into a super-healthy configuration by changing it only in the stains.  $\dashv$

Recall Definition 9.2 of admissibility.

**Definition 10.6** (Super-admissibility) A super-annotation is *super-admissible* on an interval  $K$  if it is admissible on  $K$ , further consider any interval  $J \subseteq K$  of size  $\leq QB$ .

- a3) At most 3 stains intersect  $J$ .
- a4) If  $J$  contains  $k$  stains, surrounded by some healthy area (we already know  $k \leq 3$ ), while the head is at a distance  $> 2B$  within the clean area, then the total size of these stains is at most  $k \cdot c_{\text{stain}}B$ , where

$$c_{\text{stain}} = (f_1 + 2)\beta. \quad (10.1)$$

- a5) If an outer colony in  $J$  intersects two stains then it simulates a cell state with  $Turned = 1$ .

⌋

Let us motivate requirement (a5). One stain can arise and remain somewhere for example if a burst occurs in the last sweep of a work period at the bottom of a zig. A second stain can remain at the end of a work period in which the simulated machine makes a turn. If for example the turn was to the left, then the head might not visit the end of the colony where this stain is. But this sets  $Turned \leftarrow 1$  in the simulated cell. Requirement (a5) says that in an annotated configuration, this is the only way for two stains to remain outside the workspace.

A configuration may allow several possible super-annotations; however, the valid codewords (referred to in the definition of super-health) that can be recovered from it do not depend on the choice of the annotation.

The following is an immediate consequence of the definition of rebuilding, namely the fact that in a clean area and the absence of noise, if rebuilding is needed then it will be started and will succeed.

**Lemma 10.7** *Suppose that a rebuilding procedure runs noiselessly from start to finish in a clean interval  $I$ , starting on the boundary of a subinterval  $J \subset I$  that is super-admissible at the beginning. If  $K$  denotes the output interval of rebuilding, then  $J \cup K$  will be super-admissible.*

### 10.3 The simulation codes

Let us now define formally the codes  $\varphi_{*k}, \Phi_k^*$  needed for the simulation of history  $(\eta^{k+1}, Noise^{(k+1)})$  by history  $(\eta^k, Noise^{(k)})$ . Omitting the index  $k$  we will write  $\varphi_*, \Phi^*$ . To compute the configuration encoding  $\varphi_*$  we proceed first as done in Section 6.3, using the code  $\psi_*$  there, and then add some initializations: In cells of the base colony and its left neighbor colony, the sweep and drift fields *Sweep* and *Drift* are set to  $Last(1) - 1, 1$ , and  $Last(1), 1$  respectively. In the right neighbor colony, these values are  $Last(-1)$  and  $-1$  respectively. In all other cells, these values are empty. The *Addr* fields of each colony are filled properly: the *Addr* of the  $j$  cell of a colony is  $j \bmod B^*$ .

Peter remark 6. Picture? J

The value  $Noise^{(k+1)}$  is obtained by a residue operation just as in Definition 5.2 of sparsity. It remains to define  $\eta^* = \eta^{(k+1)}$  when  $\eta = \eta^k$ . Parts of the history that are locally super-annotated will be called *clean*. In the clean part, if no colony has its starting point at  $x$  at time  $t$ , set  $\eta^*(x, t) = \text{Vac}$ . Otherwise  $\eta^*(x, t)$  will be decoded from the *Info* track of this colony, at the beginning of its work period containing time  $t$ . More precisely:

**Definition 10.8** (Scale-up) Let  $(\eta, Noise)$  be a history of  $M$ , where  $Noise = Noise^{(k)}$  as in Definition 5.2. We define  $(\eta^*, Noise^*) = \Phi^*(\eta, Noise)$  as follows. Let  $Noise^* = Noise^{(k+1)}$ . Consider position  $x$  at time  $t$ , and  $J = (t - T^*, t]$ . If  $x$  is not contained in some interval  $I$  such that  $\eta$  is super-admissible over  $I \times J$  then  $\eta^*(x, t) = \text{Bad}^*$ . Assume now that it is contained, and let  $\chi(\cdot, u)$  be some super-clean history satisfying  $\eta$  over  $I \times J$ . If  $x$  is not the start of a colony in  $\chi$  then let  $\eta^*(x, t) = \text{Vac}$ ; assume now that it is. Then let  $t' \in J$  be the starting time in  $\chi$  of the work period of  $C$  containing  $t$ , and let  $\eta^*(x, t)$  be the value decoded from  $\eta(C, t')$ . In more detail, as said at the end of Section 5.2, we apply the decoding  $\psi^*$  to the interior of the colony it to obtain  $\eta(x, t)$ . J

This definition decodes super-admissible intervals and histories for  $\eta$  into super-clean intervals and histories of  $\eta^*$ , while preserving the the property (CI) of cleanness in Definition 6.3.

## 11 Isolated bursts

Here, we will prove that the healing procedure indeed deals with isolated bursts. Our goal is to show that it provides relief, as required in an admissible annotated trajectory.

Bursts can create disorder. For its elimination we will rely on the Escape, Spill Bound and the Attack Cleaning properties of a trajectory, see Definition 6.10.

Isolated bursts don't create disorder larger than  $3\beta$ . The head can escape a disorder interval  $I$  via the Escape property; while it is inside the spreading of this interval is limited by the Spill Bound property. Every subsequent time when the head enters and exits  $I$  this gets decreased via the Attack Cleaning property, so it disappears after  $O(\beta)$  such interactions—see Lemma 11.2 below.

In a clean configuration, whenever healing started with an alarm, the procedure will be brought to its conclusion as long as no new burst occurred. The trajectory properties, however, do not allow any conclusion about the state of the cell to which the head emerges from disorder. This complicates the reasoning, and may require several restarts of the healing procedure. By design, the healing

procedure can change the *Core* track only in one cell, even if the head emerged from disorder, with wrong information. The following lemma limits even this kind of damage: it says that the head with the wrong information may increase some existing island, but will not create any new one.

**Lemma 11.1** *In the absence of noise, no new island will arise.*

*Proof.* The islands are defined only by the *Core* track. In normal mode, this track changes only at the front. If this is not the real front, then we are already in or next to an island.

The healing procedure's change of *Core* is part of a stitching operation. Looking at the different cases of the proof of Lemma 9.5, we see that inside a healthy area, healing can only change the *Core* track in two ways.

The first way is case 1, when the *Sweep* values were changed in a segment not belonging to the interior. Such an operation can be applied to any healthy configuration without affecting health.

The second case is when the front is moved left or right. This does not affect health either.  $\square$

The following lemma is central to the analysis of the behavior of the machine under the condition that bursts are isolated.

**Lemma 11.2 (Healing)** *In the absence of noise in  $M^*$ , the history can be super-annotated. Also, the decoded history  $(\eta^*, \text{Noise}^*)$  satisfies the Transition Function property of trajectories (Definition 6.10).*

*Proof.* The proof of super-annotation is by induction on time, extending the super-annotation into the future. The extension is straightforward as long as no distress event is encountered. The Transition Function property is observed, as no obstacle arises to the simulation. Stains do not cause problems: the computation stage of the simulation cycle eliminates them using the error-correcting code.

Consider now the occurrence of a distress event. This can be due to either a burst or the encounter with an island. In the latter case, before the relief a burst can still hit. In the analysis below, then we will just cut our losses and restart, knowing that burst cannot hit again before relief.

At the time of the distress event, let us draw an interval  $I$  of size  $QB$  centered around the head. The head will not leave it before relief, so we will consider the changes of the configuration inside it. Let  $S_1, \dots, S_m$  be the list of substantial segments of  $I$ , and  $A_1, \dots, A_{m-1}$  the ambiguous segments between them. Note that  $m = O(1)$ . Let  $R$  be the set of those  $i$  for which  $S_i \cup A_i \cup S_{i+1}$  is clean. For  $i \in R$  let  $n_i$  be the number of stitching steps by the algorithm of Lemma 9.5 needed to

stitch them together. Let  $N = \sum_{i \in R} n_i$ . Whenever the head enters disorder, we will mark the edge where it entered as *attacked*.

We introduce a few variables for the proof.

- $D$  = be the total size of disorder, divided by  $B$ .
- $E$  = the number of un-attacked edges of disorder where the head can exit without entering (because it is on the side of the disorder). This number never increases.
- $F$  = the number of un-attacked edges where the head cannot exit without entering (because it is away from the disorder).
- $P$  = be the number of steps that the front moves forward (including the ones after possibly changing the meaning of forward at a regular turn).

The following kinds of event may occur:

- h1) With a stitching done,  $N$  decreases by 1 while possibly moving the front backward.
- h2) The head enters disorder, decreasing  $F$ .
- h3) The head leaves disorder on a non-attacked edge, decreasing  $E$ .
- h4) The head leaves disorder on an attacked edge, decreases the disorder by at least  $B$  and increases  $F$  by 1.
- h5) A set  $A_i$  becomes clean: then  $i$  gets added to  $R$ ; this can happen only  $m$  times.
- h6)  $N$  increases in the healing mode by 1. This can only happen after the head entered the clean area in healing mode (with the wrong information). At the exit, either  $E$  or  $D$  had to decrease.
- h7) The head approaches the front: if it does not reach there then it gets closer by  $\Omega(\beta)B$ .
- h8) The front moves forward in normal mode (possibly after making a regular turn, and thus changing the forward direction). This may also increase  $N$  by 1 (and is the only way to do so), say by decreasing a segment  $S_i$ . But it is followed by zigging. The zigging may hit disorder, leading to case (h2), or find something wrong—and trigger new healing, which leads to some of the other events. Otherwise the island around the front will be deleted, and the head becomes distress-free.

The above possibilities suggest a potential function

$$N + c_D D + c_E E + c_F F + c_m m - c_P P$$

where  $c_D, \dots$  are appropriate positive constants. Let us look at what each possibility does to the potential.

Cases (h1-h3) decrease the potential if  $c_P < 1$ . Case (h4) decreases the potential if  $c_D > c_F$ . There are only  $O(1)$  cases of type (h5), increasing  $N$  by a total of  $O(\beta)$ .

In case (h6), if  $c_D$  and  $c_E$  are large enough, the increase in  $N$  can be charged to a decrease in  $D$  or  $E$ .

Case (h7) will not change the potential but there are at most  $O(1)$  consecutive such cases.

Consider case (h8). If the zigging hits new disorder and  $c_F > 1$  then the potential decreases. If it triggers new healing then this will lead to one of the other cases, compensating for the increase of  $N$  if the constants (other than  $c_P$ ) are large.

These considerations show that relief indeed follows distress in time  $O(\beta^2 T)$ . At that point a normal-mode step moving the front has been made, followed by zigging. Therefore the island causing the distress must have been eliminated, allowing the simulation to continue. The stain caused by the island remains, but as discussed after Definition 10.6, the simulation guarantees that the size and number of stains remains bounded as required by that definition. The simulation also continues to deliver the Transition Function property of trajectories.  $\square$

## 12 Cleaning

This section will scaled the Spill Bound, Escape, Attack Cleaning and Pass Cleaning properties of trajectories.

Let

$$\pi^* = \pi + \theta + 3. \quad (12.1)$$

### 12.1 Cleaning the current level

The following lemma will be used in the the scale-up of the Dwell Cleaning and Pass Cleaning properties.

Let

$$r = 4, \lambda = 3 < \pi. \quad (12.2)$$

**Lemma 12.1** *Let  $P$  be a space-time path of the head with at most  $\lambda \pi^2$  bursts occurring during the passes and intrusions of  $P$  into  $I$ . Let  $J_1 \subset I$  be an interval of size  $\theta B/2$  that  $P$  passes  $\pi + \theta$  times, at a distance at least  $r\theta B$  from the bursts of  $P$ . Then there is a time during  $P$  when  $\text{Int}(J_1, c_{\text{pass}}B)$  becomes clean.*

The proof of this lemma does not rely on any details of the simulation program.

*Proof.* Assuming the statement is not true we will arrive at a contradiction. Assume that  $J_1$  is the leftmost interval with the required properties that does not become clean at any time during  $P$ .

If during the first  $\pi$  passes all the intrusions of  $P$  into  $J_1$  are covered by fewer than  $\pi^2$  rectangles, then the Pass Cleaning property for  $M$  implies that the interior of  $J_1$  becomes clean by their end time (denote it by  $t_2 < t_1$ ). Suppose now that this is not the case, so for example the left intrusions need more than  $\pi^2/2$  rectangles to cover. Between each two passes, all but one of the rectangles are maximal, so at least

$$K_2 = \pi^2/2 - \pi$$

of them are. We will call these *maximal intrusion rectangles*.

It some interval of size  $2B$  within distance  $\theta B/2$  of intersecting  $J_1$  becomes clean by time  $t_1$  then the Attack property will extend this interval to the right in the remaining  $\theta$  passes after  $t_1$  and clean  $J_1$  as well.

**Peter remark 7. Elaborate!** ┘

But we assumed this does not happen. If one of the rectangles intersecting  $J_1$  is max height, then the Dwell Cleaning property implies an interval of size  $2B$  in it that becomes clean before time  $t_2$ , which again leads to contradiction by the Attack property and the remaining  $\theta$  passes. So all the  $K_2$  rectangles have maximum width. Let  $J_2$  be the interval of length  $\theta B/2$  adjacent to  $J_1$  on the left. Since the intrusion rectangles don't reach the right end of  $I$ , they extend to a distance  $\theta B/2$  to the left of  $I$ , so each of them passes  $J_2$ .

Let us iterate this reasoning for  $i = 2, \dots, r$ . Assume we have  $K_i$  noise-free passes over  $J_i$  before time  $t_i$ . We can divide the passes before the the time  $t_{i+1} < t_i$  at the beginning of the last  $\theta$  of them into groups of  $\pi$  consecutive passes, so we have now at least

$$K_i - \theta - \pi$$

passes over  $J_i$  before  $t_{i+1}$  divided into consecutive groups of size  $\pi$ . Each group fails to clean  $J_i$ , so there are at least  $\pi^2$  intrusions per group, altogether at least  $\pi(K_i - \theta - \pi)$  intrusions. Without loss of generality assume that at least half of these are from the left. Let  $J_{i+1}$  be the interval of size  $|J_i|$  adjacent to  $J_i$  on the left. If  $J_{i+1}$  becomes clean before  $t_{i+1}$  then by the Attack property  $J_i$  would become clean by time  $t_i$  contrary to the assumption.

Just as in case of  $i = 1$ , between each pair of passes, all but one of the intrusions has a maximum-width covering rectangle. This gives

$$K_{i+1} = \pi(K_i - \theta - \pi)/2 = K_i(\pi/2) - (\theta + \pi)\pi/2 \quad (12.3)$$



passes over  $J_{i+1}$ . Hence for  $i \geq 3$

$$K_i = (\pi/2)^{i-1}\pi - (\theta + \pi)((\pi/2) + (\pi/2)^2 + \cdots + (\pi/2)^{i-2}) \geq (\pi/2)^i$$

if  $\pi$  is large.

The situation changes once  $J_i$  is such that bursts can happen in it, which can happen if  $i \geq r$ , so we will handle the case  $i = r$  differently. By the assumption the total number of bursts is at most  $s = \lambda\pi^2$ . Each burst may affect at most one pass, so there remain at least  $K_i - s$  burst-free ones of them, separated into at most  $s + 1$  burst-free groups. Given the size of  $\pi$ , at least one of these groups has size at least

$$K_{r+1} = \frac{K_r - s}{s + 1} > \frac{K_r}{2\lambda\pi^2} \geq \pi^{r-2}2^{-r-1}/\lambda.$$

Let  $t_{r+1}$  be the endtime of the last pass in this group. With  $r = 4$  in (12.2) this gives  $K_5 \geq \pi^2/2^5\lambda > \pi$ . From here on we can again use (12.3), since no burst appears at all during the time of this group of passes. But since  $P$  is finite, sooner or later this process must terminate, leading to a contradiction.  $\square$

## 12.2 Escape

This section scales up the Escape property of trajectories.

**Definition 12.2** A maximal clean interval will be called a *clean hole*. Let  $\mathcal{K}(t)$  denote set of all clean holes of size  $\geq c_{\text{attack}}B$  at time  $t$ , let  $K(t) = \bigcup \mathcal{K}(t)$ . Let  $\mathcal{K}^*(t)$  be the set of those super-healthy clean holes, and  $K^*(t)$  be their union.  $\dashv$

Of course,  $K^*(t) \subseteq K(t)$ .

Consider a burst-free space-time set  $I \times J$ : since no disorder can appear during it in the interior of a clean interval, if  $I_1$  is a clean interval at time  $t$  and  $I_2, I_3$  are clean intervals at time  $t + 1$  intersecting  $I$  then the smallest interval containing  $I_2 \cup I_3$  is also clean at time  $t + 1$ . So the elements of the set  $\mathcal{K}(t)$  of clean holes of size  $\geq c_{\text{attack}}B$  will not break up: they can grow, shrink but only slightly, bounded by the Spill condition (and thus will not disappear) or merge; new elements can appear as well. We can similarly follow a single clean hole  $I(t)$ .

Just as a clean interval cannot break up in the absence of bursts, a super-healthy interval cannot break up either. So we can follow the evolution of super-healthy clean holes in  $\mathcal{K}^*(t)$  similarly. Recall that a super-healthy interval in history  $\eta$  is clean in the decoded history  $\eta^*$ , so the simulation of  $\eta^*$  will proceed in it.

In what follows we will estimate how much the area of the sets  $K(t)$  and  $K^*(t)$  will grow with the cumulative time spent  $K(t)$ .

The following lemma considers the activity of the head while it is inside a clean hole  $I(t)$ . the case that the interval in question is clean.

**Lemma 12.3** *Let  $I(t)$  be a clean hole having size  $\leq nB$  with  $n \leq \lambda Q$ ,  $\lambda \leq 3\beta$  at the beginning of a burst-free time interval, and  $k_I(t) = |K^*(t) \cap I(t)|$ . If  $n < Q/2$  then the head will leave  $I(t)$  within time  $c_{\text{esc}}ZnT$ . Otherwise, every time interval of its stay that is longer than  $c_{\text{esc}}\lambda T^*$  grows  $k_I(t)$  by at least  $QB$ .*

*Proof.* We will see that every certain amount of time some kind of progress is made.

1. Suppose that we are at some time when the rebuild procedure had started, from a base of rebuild-marked cells large enough not to result in alarm (renewed call for healing) after the first zigging. Then the rebuilding will be completed.
2. Suppose that we are at some time when the mode is normal. Then within  $Z$  steps either a healing call happens, or a step of progress will be made in the ordinary work of simulation.
3. Suppose that at some time healing will be called. Then according to the proof of Lemma 11.2, we either arrive at case 1, or in  $O(\beta^2 < Z)$  steps or healing finishes in normal mode with at least one step made in the ordinary work of simulation.

Indeed if healing succeeds it leaves a healthy area. Lemmas 8.8 and 9.6 imply that the overlapping successful healing areas can be combined, so healing will not be repeated over the same area, and thus can slow down progress only by a factor  $O(\beta^2)$  (negligible compared to the  $Z$  times slowdown by zigging), and even this in at most one sweep of simulation over any area.

4. Suppose that  $n > QB$  and the head stays longer than  $c_{\text{esc}}\lambda T^*$ . will grow by at least  $QB$ . In this much time, if any rebuilding starts then it will finish and extend  $K^*(t)$  by a colony. Without rebuilding, the continued healings add some colonies to the health area; any such colony will be turned super-healthy by the first complete work-period of the simulation. If the head is already in a super-healthy area then it will perform the simulation of  $M^*$ . The simulated machine  $M^*$  has the same program, so it will also make switchbacks of at least  $E > 3\beta$  steps (for healing) or even the bigger ones for zigging. This extends the super-healthy area unless interrupted by rebuilding, which as seen extends it also.

□

**Definition 12.4** Let  $I$  be a clean interval, with the head on its right. We will say that its right end is *directed*, if the right end, except for a final interval of size  $2E$ , is covered by a pair of consecutive intervals  $K \cup L$  where  $K$  is a super-healthy interval containing at least two complete colonies,  $L$  is covered by cells marked for

rebuilding and has size  $< 4QB$ , and the rightmost colony of  $K$  is in the process of transfer to the right. Directedness of the left end is defined similarly.  $\lrcorner$

**Lemma 12.5** *Let  $I(t)$  be a clean hole having size  $\leq nB$  with  $n \leq \lambda Q$ ,  $\lambda \leq 3\beta$ , during a burst-free time period.*

- a) *If the head stays in  $I(t)$  longer than  $c_{\text{esc}}ZnT$  and  $|I(t)|$  did not increase by more than  $QB$ , then the end on which it leaves becomes directed.*
- b) *After consecutive passings of the head into and out of a directed end, this end stays directed. If after entering, the head spends a cumulative time of  $2T^*$  in  $I(t)$ , the value  $k_I(t) + |I(t)|$  increases by at least  $QB$ .*

*Proof.* To (a): In a clean interval, under burst-free conditions, the head is preparing a healthy area either by successive healings or by rebuilding. The time spent in  $I(t)$  without increasing it significantly indicates that a simulation must have started in a healthy pair of colonies. It can only get interrupted during transfer, either when the head leaves  $I(t)$  or when healing/rebuilding starts. The rebuilding can also only be interrupted when the head leaves  $I(t)$ .

To (b): Assume that the head enters from the right. If it starts healing or rebuilding then it does not alter the directedness, since the rebuilding does not encompass both healthy colonies of the pair doing the transfer. The head cannot get past the left healthy colony without either completing its right transfer operation or completing a rebuilding action. If the transfer or rebuilding finishes then a colony is added to  $K^*(t)$ , thus increasing it by  $QB$ . The cumulative time for this is at most  $T^*$ . If rebuilding keeps restarting and never finishes during the time  $2T^*$  then during the time  $T^*$  spent on these attempts, given that rebuilding takes only  $O(ZQT)$  time, the head must have passed the right end so many times that the Attack property increases  $|I(t)|$  by  $QB$ .  $\square$

The next lemma considers a situation similar to Lemma 12.3, but allowing multiple entries and exits.

**Lemma 12.6** (Expand hole) *Let  $I(t)$  be a clean hole having size  $\lambda QB$  with  $1/2 \leq \lambda \leq 3\beta$  at the beginning of a burst-free time interval. There is a constant  $c$  with the following property. If the head spends cumulative time  $c\lambda T^*$  in  $I(t)$  then at the end of  $J$  the hole grows in size by at least  $QB/2 - c_{\text{spill}}B > QB/3$ .*

*Proof.* If the head spends time  $\leq cZQT$  in  $I(t)$ , the constant  $c$  from Lemma 12.5, then we say that its stay was *short*. If it spends time  $\geq c_{\text{esc}}\lambda T^*$  in  $I(t)$  (from Lemma 12.3), then we will say that the stay was *long*. Otherwise we will say that the stay was *medium*.

1. After a short stay,  $I(t)$  expands by at least  $B/2$  via the Attack property.

2. After a long stay (even without leaving), by Lemma 12.3, the value of  $k_I(t) = |K^*(t) \cap I(t)|$  increases by at least  $QB$ .
  3. Lemma 12.5 implies that after a medium stay and further cumulative time of  $2T^*$  inside  $I(t)$ , the value  $s(t) + |I(t)|$  increases by at least  $QB$ .
- Adding up these possibilities will complete the proof.  $\square$

The following lemma is the scale-up of the Escape condition.

**Lemma 12.7** (Escape) *In the absence of Noise\*, the head will leave any interval  $G$  of size  $nB$  with  $n \leq \lambda Q$ ,  $1 \leq \lambda \leq 3\beta$ , within time  $c_{\text{esc}}\lambda T^*$ .*

*Proof.* If  $J$  is the time interval considered, then we can ignore the at most one burst happening during  $J$  by considering the half of  $J$  where it does not occur.

At any time  $t$ , we will partition the interval  $G$  into subintervals as follows. First, let  $\mathcal{K}(t)$  be the set of clean holes of size  $\geq c_{\text{attack}}B$  inside  $G$ , and  $K(t)$  their union. Each interval between two such clean holes that is larger than  $\theta B/2$  will be subdivided into a sequence of subintervals; the first and last one have size  $\leq \theta B/2$ , the other ones have size  $\theta B/2$ . Let us call all members of this partition of  $G$  *blocks*; the clean holes among them will be called *clean*, the other ones are called *short*.

As time passes new clean blocks may arise, the existing ones may increase, and neighboring ones may merge. (They may also decrease due to spill of disorder but this decrease is temporary until the next increase due to attack.)

We will consider a sequence of times  $t_1 < t_2 < \dots$  defined as follows.  $t_1$  is our starting time. If  $t_i$  is defined and at time  $t_i$  the head is in some block  $S$  then  $t_{i+1}$  is the first time after  $t > t_i$  at which

- if  $S$  is clean then the head is either not in  $S$  or  $t = t_i + T$ ;
- if  $S$  is short then the head is not in either  $S$  or in any short block next to  $S$ .

The Escape property of trajectories implies  $t_{i+1} - t_i \leq e_1 T$ , where

$$e_1 = c_{\text{esc}} \cdot 3\theta/2,$$

since the length of a union of three consecutive short blocks is  $(3\theta/2)B$ . By definition, between times  $t_i$  and  $t_{i+1}$  one of the following events occurs. We will show the contribution to the growth of  $|K(t)|$  made by them. Clearly  $K(t)$  cannot grow larger than  $\lambda QB = |G|$ .

- e1) A boundary is crossed between clean and short blocks. A pair of passes over the same boundary results in an increase of  $K(t)$  by  $B$ . The number of unpaired passes over boundaries is at most the upper bound of the number of such boundaries, that is  $e_2 Q$  where

$$e_2 = \lambda / c_{\text{attack}}.$$

So if  $t$  is the time spent on events of this type then there are at least  $t/e_1T$  such events, and then the growth of  $K(t)$  is at least

$$\frac{(t/e_1T - e_2Q)}{2} \cdot \frac{B}{2}.$$

Assuming  $t \geq 2e_1e_2QT$  this is at least  $tB/8T$ . This becomes larger than  $|G| = \lambda QB$  if  $t > 8\lambda QT$  so the time spent on these events is less than  $8\lambda QT \leq T^*$ .

- e2) Two clean blocks merge. The number of these events is limited by  $e_2Q$ , for a total time limited by  $e_1e_2QT \leq T^*$ .
- e3) The cumulative time spent in  $K(t)$  increases by  $T$ . Let  $c_{\text{expand}}$  be the constant  $c$  of Lemma 12.6. By that lemma, if the head spends time  $c_{\text{expand}}\lambda T^*$  on these events then  $K(t)$  increases by at least  $QB/3$ . So it cannot spend more than  $3c_{\text{expand}}\lambda^2T^*$  without escaping  $G$ .
- e4) The number of passes over a short block  $S$  increases by 1. If  $\pi + \theta$  such events occur over a short block  $S$  then Lemma 12.1 becomes applicable (with no bursts!) and  $\text{Int}(S, c_{\text{pass}}B)$ , of size  $\geq \theta B/6$ , becomes clean, increasing  $|K(t)|$  by  $\theta B/6$ . If time  $t$  is spent on such event, then the increase is at least

$$\frac{t\theta B}{6e_1T(\pi + \theta)}.$$

This becomes larger than  $\lambda QB$  if  $t > 6e_1T(\pi + \theta)Q$ . But  $T^*/T > 6(\pi + \theta)Q$ , so the total time spent on these events is at most  $T^*$ .

□

### 12.3 Pass cleaning

The following lemma states the Pass Cleaning property for machine  $M^*$ .

**Lemma 12.8** *Suppose that a path  $P$  has no bursts of  $\eta^*$ , it passes  $\pi^*$  times the interval  $I$ , and there is a  $(\theta QB, \theta UT)$ -covering of size  $\leq (\pi^*)^3$  for  $I$  and  $P$ . Then by end of the  $\pi^*$ th pass the interior  $\text{Int}(I, 5QB)$  becomes clean for  $\eta^*$ .*

Because  $\theta \leq \gamma$  and the path  $P$  contains no burst of  $\eta^*$ , each rectangle of the  $(\theta QB, \theta UT)$ -covering contains at most one burst; so the number of bursts during all the intrusions is at most  $(\pi^*)^2$ .

The first part of the proof shows that after  $\pi + \theta$  passes, the interior of  $I$  becomes clean, except for  $O(\pi^2)$  islands. The next part shows that the additional passes build up the next level (three should suffice).

### 12.3.1 Building up the next level

We will show that if an interval  $I$  of size  $\theta QB$  is clean except for possibly

$$\kappa = (2\lambda + 1)(\pi + \theta + 3) = O(\pi^2) \quad (12.4)$$

islands then in a constant number of passes it will be populated with healthy colonies, provided no more than  $\kappa$  bursts happen in it during and between these passes. Note that for restoring the next-level structure it is allowed to destroy some existing colonies.

**Definition 12.9** An interval  $J$  not containing the head is called *directed* if its cells, possibly with the exception of an island, point towards some possible front (in normal operation or rebuilding). This front can be inside or outside  $J$ . If it is to the left of  $J$  then the head must also be on the left of  $J$ , similarly for right. If it is inside then the head must be inside, with one exception: if, say, the front is within  $Z$  steps to the left of  $J$  and moving right then the head may be on the left of  $J$  (say, captured there while zigging).  $\lrcorner$

For the lemma below, recall that  $E$  is the maximum number of cells in a healing area. It relies on the fact that  $E$  is much smaller than the size  $Z$  of zigging.

**Lemma 12.10** *Suppose that  $J$  is directed, the head enters  $J$  and leaves it. Let  $J'$  be  $J$  if the intrusion was noiseless, and  $\text{Int}(J, EB)$  if it contains a burst. Then  $J'$  will be also directed after the intrusion.*

*Proof.* In normal operation or rebuilding, the head moves the front with itself and returns to it from every zig, except when it is captured at an end of  $J$  (by a burst or disorder). If an island is encountered or a burst occurs then it will be healed, except when the healing interval (whose maximum size is  $EB$ ) intersects the boundary of  $J$  (where the disorder may capture the head). The decrease of  $J$  to  $J'$  reflects this.  $\square$

**Lemma 12.11** *Consider four passes, starting with a right pass, over a clean interval  $J$  that gets no burst during this time. Let  $J'$  be an interval adjacent to  $J$  on the right that is smaller than  $ZB - |J|$ , and the total amount of disorder that may be created in it during all the passes and intrusions considered is less than  $|J|/2$ . Then after one of the two left passes,  $J \cup J'$  becomes left-directed.*

*Proof.* Suppose after a right pass all cells of  $J$  are directed to the right. Then on the next left pass, disorder on the right within distance  $ZB - |J|$  will be hit by every right zig, which eliminates a  $B/2$ -size piece of it by the Attack property. Because of our condition that the amount of disorder in  $J'$  is at most  $|J|/2$ , there will be enough right zigs to eliminate all disorder.

It can only happen that not all cells of  $J$  are directed right after the first right pass if at the end of the pass the front makes a left turn before reaching the right end of  $J$ , and the head reaches it only during a right zig. But then the feathering property implies that during the following rightward pass it cannot make a left turn within  $ZB$  of the right end of  $J$ , therefore  $J$  will become directed right.  $\square$

**Lemma 12.12** *Consider an interval  $K$  of size  $\leq 3\kappa EB$ , in  $\text{Int}(I, 2\kappa^2 B)$ . Within four passes (starting from the left) it becomes left-directed.*

*Proof.* (The proof assumes  $Z > 3\kappa^2$ .) Since there are at most  $\kappa$  islands (of size  $\leq \beta B$ ), there is a clean interval  $J$  in  $I$  of size  $\kappa B$  on the left of  $K$  at a distance at most  $\kappa(\kappa + \beta)B < (2\kappa^2 - \kappa E)B$ . Let  $J'$  be the smallest interval adjacent to  $J$  containing  $K$ . Lemma 12.11 implies that  $J'$  becomes left-directed during one of the two left passes.  $\square$

**Lemma 12.13** *After 4 passes, the interval  $\text{Int}(I, 3\kappa^2 B)$  will be directed.*

*Proof.* Consider a subinterval  $J$  of  $\text{Int}(I, 2\kappa^2 B)$ , of size  $\kappa EB$ . Let  $J'$  be the interval of size  $3\kappa EB$  for which  $J = \text{Int}(J', \kappa EB)$ . By Lemma 12.12,  $J'$  becomes left-directed within four passes. After this, Lemma 12.10 makes sure that besides an island created by the last burst,  $J'$  can get islands only at its edges. These can decrease  $J'$  by at most  $EB$  at a time. There are at most  $\kappa$  such times, so they never reach  $J$ . The fourth pass cleans away all islands but the one created by a burst within the pass itself.  $\square$

Now the next pass is going through the directed interval  $\text{Int}(I, 3\kappa^2 B)$ . If it does not already find healthy colonies it will apply the rebuilding procedure to produce them; nothing prevents this in the directed interval.

## 12.4 Attack cleaning

Eventually, we will prove the Attack Cleaning property of trajectories (Definition 6.10) for the decoded history  $(\eta^*, \text{Noise}^*)$ . However, first we prove a weaker version, requiring the space-time rectangle of interest to be free not only of  $\text{Noise}^*$  but also of  $\text{Noise}$ —that is burst-free.

**Lemma 12.14** *Consider a trajectory  $\eta$  in a noise-free space-time rectangle. Here, the decoded history  $\eta^*$  satisfies the Attack Cleaning property.*

*Proof.* The property says the following for the present case. For current colony-pair  $x, x'$  (where  $x' < x + 2QB$ ), suppose that the interval  $[x - c_{\text{attack}}QB, x' + QB)$  is clean for  $M^*$ . Suppose further that the transition function, applied to  $\eta^*(x, t)$ , directs the head right. Then by the time the head comes back to  $x - c_{\text{attack}}QB$ , the

right end of the interval clean in  $M^*$  containing  $x$  advances to the right by at least  $QB$ .

The computation phase of the simulation on the colony pair  $x, x'$  is completed without the disturbing effect of noise: even the zigging does not go beyond the boundary. Then the transfer phase begins which enters the disorder to the right of  $x' + QB$ .

We argue that there are only two ways for the head to get back to  $x - c_{\text{attack}}QB$ .

- at1) The transfer into a new colony pair with starting point  $y \geq x' + QB$  succeeds despite the disorder, and the clean interval extends over it, before the head moves left to  $x - c_{\text{attack}}QB$  in the course of the regular simulation. Some inconsistencies may be discovered along the way, but they are corrected by healing.
- at2) The inconsistencies encountered along the way trigger some rebuilding processes. Eventually, a complete, clean rebuilding area is created, the rebuilding succeeds, leaving a clean colony also to the right of  $x' + QB$ .

The Spill Bound property guarantees that the area to the left of  $x' + (Q - c_{\text{spill}})B$  remains clean, therefore the only way for the head to move left of that is by the rules. Suppose that rebuilding is not initiated (with creating a substantial germ): then moving left can only happen by the normal course of simulation: the transfer stage of the simulation must be carried out, and this requires at least as many attacks to the right as the number of sweeps in the transfer stage. Every attack (followed by return) extends the clean interval further, until the whole target colony becomes clean, and the transfer completed. This is the case (at1).

Recall the rebuilding procedure in Section 9.3: the rebuilding area extends  $2.5Q$  cells to the left and right from its initiating cell. This may become as large as  $5QB$  to the left and right. If rebuilding is initiated, its starting position is necessarily to the right of  $z = x' + (Q - \text{PadLen} - 2)B$ . It then may extend to the left to at most  $z - 5QB$  (this is overcounting, since the cells of the colony of  $x$  are all adjacent). Its many sweeps will result in attacks that clean an area to the right of the starting point. The procedure may be restarted several times, but those restartings will also be initiated to the right of  $z$ . Therefore the rebuilding area does not extend to the left of  $z - 5QB$ : if the head moves to the left of this, then the rebuilding must have succeeded. The rebuilding also must find or create a colony manifestly to the right of the restarting site: this will be to the right of  $z$ , moving this way the boundary of the area clean in  $M^*$  by at least  $QB$ .  $\square$

Note that in the process described in the above proof, it is possible that the rebuilding finds a competing colony  $C$  starting at some  $y \in x' + QB + [-\beta B, 0)$  which slightly (by the size of an island) overlaps from the right with the colony of



$x$ . The rebuilding may decide to keep  $C$  and to overwrite the rest of the colony of  $x'$  as a bridge (or even target). This does not affect the result.

## 13 After a large burst

Our goal is to show that the simulation  $M \rightarrow M^*$  defined in Section 10.3 is indeed a simulation. Section 11 shows this as long as the head operates in an area that is clean for the simulated machine  $M^*$  (can be super-annotated), and has no noise for machine  $M^*$  (that is its bursts on the level of machine  $M$  are isolated). In other words, essentially the Transition Function property of Definition 6.10 of trajectories for the simulated machine  $M^*$  has been taken care of.

The new element is the possibility of large areas that cannot be super-annotated: they may not even be clean, even on the level of machine  $M$ . The Spill Bound, Attack Cleaning, Dwell Cleaning and Pass Cleaning properties still must be proven.

### 13.1 Spill bound

One of the most complex analyses of this work is the proof that the simulated machine  $M^*$  also obeys the Spill Bound. Let us outline the problem.

We are looking at the boundary  $z$  of a large area that is clean for the machine  $M^*$ : without loss of generality suppose that this is the right boundary. We will be looking at it in a space-time rectangle  $[a, b) \times [u, v)$  that is noise-free in  $M^*$ . The interesting case has  $a < z < b$  with  $z - a, b - z = O(QB)$ . The assumption allows occasional bursts of noise of the trajectory  $\eta$  of  $M$ , but no two of these bursts must occur in a space-time rectangle of size comparable to the size of a colony work period of  $M$ . The Spill Bound for trajectory  $\eta$  keeps the disorder of  $\eta$  within  $O(B)$  on the left of  $z$  while  $\eta$  is noise-free. If we could assume  $\eta$  noise-free then the heal/rebuild procedures would also keep the disorder of  $\eta^*$  from spilling over by more than  $O(B^*) = O(QB)$ . However, nothing is assumed about the length of the time interval  $[u, v)$ . If the head would spend all the time within  $O(QB)$  of  $z$  then due to the Dwell Cleaning property of trajectories, the area would be cleaned out, again preventing spilling. But the head can slide out far to the right of  $b$  fast, since the disordered area on the right of  $z$  is arbitrarily large. Then it can come back much later to the left of  $z$ , and by a burst (allowed since much time has passed) can deposit an island of disorder there. Repeating this process would produce unlimited spillover, not only of the disorder of  $\eta^*$  but even that of  $\eta$ .

This is where the Pass Cleaning property helps. If the above process is repeated  $\pi$  times, the  $\pi$  passes would clean out an interval on the right of  $z$  whose size is of the order of  $QB$ , while depositing only  $\pi$  islands of disorder to the left

of  $z$ . Our construction will have  $\pi \ll Q$ : more precisely, in our hierarchy of generalized Turing machines we will have  $\pi_k = k$ ,  $Q_k = k^2$ . And  $\ll Q$  bursts will still be handled by healing/rebuilding in  $\eta$ .

Of course this sketch is very crude, but it should help motivate the reasoning that follows.

## Bibliography

- [1] Ilir Çapuni and Peter Gács. A Turing machine resisting isolated bursts of faults. *Chicago Journal of Theoretical Computer Science*, 2013. See also in arXiv:1203.1335. Extended abstract appeared in SOFSEM 2012. [2](#), [10.1](#)
- [2] Bruno Durand, Andrei E. Romashchenko, and Alexander Kh. Shen. Fixed-point tile sets and their applications. *Journal of Computer and System Sciences*, 78:731–764, 2012. [2.2](#)
- [3] Peter Gács. Reliable computation with cellular automata. *Journal of Computer System Science*, 32(1):15–78, February 1986. Conference version at STOC’ 83. [2.2](#)
- [4] Peter Gács. Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103(1/2):45–267, April 2001. See also arXiv:math/0003117 [math.PR] and the proceedings of STOC ’97. [1.4](#), [2.2](#)
- [5] G. L. Kurdyumov. An example of a nonergodic homogenous one-dimensional random medium with positive transition probabilities. *Soviet Mathematics Doklady*, 19(1):211–214, 1978. [2.2](#)